Linköping University

End of degree project

TELECOMUNICATION ENGINEERING

DVB-T PILOT SENSING ALGORITHMS

Linköping University

Department of Electrical Engineering, Communication System Division

Student: David Cordero Díaz

Supervised by Danyo Danev

Linköping, June 2010

Kommunikationssystem, ISY. Linköpings universitet. 581 83 LINKÖPING. Tel: 013-28 10 00. Fax: 013-13 92 82

Abstract

The world nowadays is in the middle of a digital to analog TV transition where digital transmission technology is substituting completely the analog television. This is because digital transmission has many advantages in terms of spectral efficiency, flexibility and robustness which make it especially attractive over analog transmission.

Digital Video Broadcasting for Terrestrial (DVB-T) is the DVB European-based consortium standard for the broadcast transmission of digital terrestrial television. This system transmits compressed digital audio, video and other data over an MPEG-2 stream, using COFDM modulation.

This paper deals with a C++ application currently being developed to simulate a DVB-T transmission in different real channels in order to test several pilot sensing algorithms, compare them and determine in which conditions and scenarios their use is more appropriated.

Therefore an emitted DVB-T signal has been generated and detected after crossing a channel. Channel models and detection algorithms have been chosen from existing ones. All the blocs are implemented as specified in the DVB-T standard (ETSI EN 300 744 V1.6.1).

Throughout this paper performance of DVB-T is defined and all the technical details needed to understand the application are explained. The application developed and its functions are described. Finally, the obtained results are evaluated and discussed.

Contents

List of figures

List of tables

Symbols

$\propto$	Minimum distance separating two constellation points
BW	Bandwidth
$c_{m,l,k}$	Complex cell for frame m in OFDM symbol l at carrier k
$cr1, cr2$	Code rate 1 and 2, for hierarchical transmission
Δ	Time duration of the guard interval
D_{lk}	DFT of transmitted OFDM symbol
f_c	Centre frequency of the emitted signal
H_0	Null Hypothesis, DVB-T signal absent
H_1	Alternative Hypothesis, DVB-T signal existing
h_{ln}	Transfer function of the channel
k	Carrier number index in each OFDM symbol
K	Number of active carriers in the OFDM symbol
K_{min}	Carrier number of the lower active carrier respectively in the OFDM signal
K_{max}	Carrier number of the largest active carrier respectively in the OFDM signal
l	OFDM symbol number index in an OFDM frame
N	Number of samples
$n(k)$	Sampled noise
$n(t)$	Continuous noise
m	OFDM frame number index
$m(k)$	DVB-T pilot signal (sampled)
$r(k)$	DVB-T received signal (sampled)
R_{lk}	DFT of the received OFDM symbol (DFT of r_{ln})
r_{ln}	Received OFDM symbol
$s(k)$	DVB-T sampled emitted signal
$s(t)$	DVB-T continuous emitted signal
T	Elementary Time period
T_s	Duration of an OFDM symbol
T_F	Time duration of a frame
T_u	Time duration of the useful (orthogonal) part of a symbol, without the guard interval
w_k	Value of reference PRBS sequence applicable to carrier k
z	Complex modulation symbol
*	Complex conjugate

Abbreviations

ATSC	Advanced Television System Committer
AWGN	Additive White Gaussian Noise
BCH	Bose - Chaudhuri - Hocquenghem code
BER	Bit Error Ratio
CP	Cyclic Prefix
COFDM	Code Orthogonal Frequency Division Multiplexing
DAC	Digital-to-analogue converter,
DFT	Discrete Fourier Transform
DMB-T/H	Digital Multimedia Broadcast-Terrestrial/Handheld
DTT	Digital Terrestrial Television
DVB	Digital Video Broadcasting
DVB-C	Digital Video Broadcasting-Cable
DVB-H	Digital Video Broadcasting-Handheld
DVB-S	Digital Video Broadcasting-Satellite
DVB-T	Digital Video Broadcasting-Terrestrial
EPG	Electronic program guide
FFT	Fast Fourier Transform
HDTV	High Definition Television
HP	High Priority, referred to hierarchy
ICI	Inter Canal Interference
IDFT	Inverse Digital Fourier Transform
IFFT	Inverse Fast Fourier Transform
ISI	Inter Symbol Interference
ISO/IEC	International Organization for Standardization / International Electronic Commission
GI	Guard Interval
ISDB-T	Integrated Services Digital Broadcasting-Terrestrial
LP	Low Priority, referred to hierarchy.
MPEG	Moving Picture Experts Group
MPEG-PS	MPEG program streams
MPEG-TS	MPEG transport streams
MUX	Multiplexer
OFDM	Orthogonal Frequency Division Multiplexing

PRBS	Pseudo-Random Binary Sequence
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quaternary Phase Shift Keying
RF	Radio Frequency
SFN	Single Frequency Network
SNR	Signal Noise Rejection
TPS	Transmission Parameter Signalling
TV	Television
UHF	Ultra-High Frequency
VHF	Very-High Frequency
VSB	Vestigial Side Band

References

Publications

[1] ETSI EN 300 744. "Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television". V1.6.1 (2009-01).

[2] IEEE 802.22-05/0263r0, "Sensing Scheme for DVB-T". Contribution for IEEE 802.22 WRAN Systems (2006-11-10).

[3] D. Landstrom, S. K. Wilson, J. – J. Van de Beek, Per Odling, and P. O. Borjesson, "Synchronization for a DVB-T receiver in presence of co-channel interference", PIMRC, IEEE, vol. 5, pp. 2307-2311, Sept. 2002.

[4] Sachin Chaudhari, Visa Koivunen and H. Vincent Poor, "Autocorrelation-Based Decentralized Sequential Detection of OFDM Signals in Cognitive Radios". IEEE TRANSACTIONS ON SIGNAL PROCESSING, vol. 57, No. 7, July 2009.

[5] Stephen J. Shellhammer, Sai Shankar N, Rahul Tandra, James Tomcik, "Performance of Power Detector Sensors of DTV Signals in IEEE 802.22 WRANs".

[6] Paul Bustamante, Iker Aguinaga, Miguel Aybar, Luis Olaizola, Iñigo Lazacano, "Aprenda C++ Básico como si estuviera en primero" (2004–04).

[7] Stormy Attaway, "MATLAB: A practical introduction to programming and problem solving", 2009.

Web Pages

[8] http://www.dvb.org/

[9] http://www.cplusplus.com/doc/tutorial/

[10] http://itpp.sourceforge.net/

Chapter 1

Introduction

1.1. What is DVB-T?

The DVB digital video broadcasting is an initiative to the market for global standardization of digital video broadcasting. Since September 1993 it has produced many specifications for terrestrial, cable and satellite TV. The result is a complete family of standards, the most representatives being: DVB-S for satellite, DVB-C for cable and DVB-T for terrestrial.

Therefore, DVB-T (Digital Video Broadcasting for Terrestrial) is the DVB European standard for the broadcast transmission of digital terrestrial television. This system transmits audio, video and data over an MPEG-2 stream, using COFDM modulation which is a complex technique of modulation bandwidth used to transmit digital information through a communications channel.

The importance of COFDM modulation is that it allows the introduction of more digital television programs in a limited number of available channels. The duration of the bits is higher than the delays, avoiding echoes and allowing the reuse of the same frequencies in neighboring antennas.

1.2. Why DVB-T should be studied?

The world nowadays is in the middle of a digital to analog TV transition, analog television is gradually disappearing and modern digital transmission technology is being used in entirely new applications and in many areas. From 1997 until now over 100 countries committed to DVB-T for Digital Terrestrial TV.

The main problem of analog TV is that it doesn't take advantage of the fact that signals vary little when moving from one picture element (pixel) to its adjacent, while there exist really a clear dependency between them. The result is a waste of the electromagnetic spectrum. In addition, because of the increasing number of broadcasting stations, interference starts to become a serious problem.

The radio channels used in digital television occupy the same bandwidth (8 MHz) as the channels used in analog television, but the use of compression techniques for audio and video signals (MPEG), allow the emission of a variable number of TV programs. Depending on the desired transmission

speed it can vary from one program of HDTV (high quality picture and sound) to five programs with similar quality to the current.

Furthermore, the use of digital terrestrial television provides a great number of advantages and new possibilities. Between them, these are the most important.

- Simplicity of installation of the receiver that use the existing antennas.
- Use of single frequency networks, so a smaller number of frequencies are required.
- Extend the available supply in number of program channels with respect to the current analog TV, allowing multiple programs and multimedia services in each radio channel.
- Digital signals are more robust regarding noise, interference and multi-path propagation. This allows an improvement in recieved image and sound quality (avoiding the effects of snow and dual analog TV picture).
- Requires less transmission power.
- Value added services such as EPG (electronic program guide), advanced teletext, video on demand, pay per view, datacasting, electronic mail, stock quotes, video telephony, home banking, home shop, etc.
- Allows portable and mobile reception.
- Allows increasing the aspect ratio. The conventional format is 4:3, whereas digital television allows a 16:9 widescreen format.
- Provides services at national, regional and local levels.

1.3. Overview of Digital TV Broadcasting

Since the implementation of digital television (DTT), different standards for Digital TV Broadcasting of terrestrial have been proposed and adopted over the world. These are ATSC (Advanced Television System Committer, USA), DVB-T (Digital video Broadcasting-Terrestrial, Europe), ISDB-T (Integrated Services Digital Broadcasting-Terrestrial, Japan) and DMB-T/H (Digital Multimedia Broadcast-Terrestrial/Handheld, Republic of China). Besides terrestrial broadcasting, video signal can be transmitted through satellite (DVB-S) and cable (DVB-C). A map showing the use of each of these standards is illustrated below:

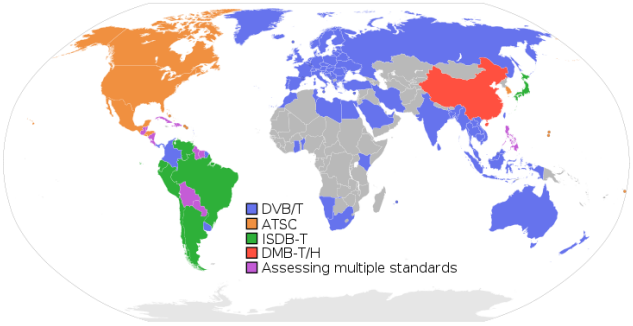

Figure 1.1 Digital broadcast standards.

A brief comparison between them is shown in the next table:

Standard	DVB-T	DVB-S	DVB-C	ATSC	ISDB-T	DMB-T/H
Modulation	OFDM	QPSK	QAM	VSB	OFDM	TDS-OFDM
Bandwidth	6 / 7 / 8 MHz	26 - 54 MHz	2 – 7.92 MHz	6 MHz	6 MHz	6 / 7 / 8 MHz

Table 1.1 Comparison of different broadcasting standards

1.4. Project Organization

The rest of this document is organized as follows.

- Explanation of the basis of DVB-T and technical details. DVB-T Standard is introduced, describing the concept of OFDM, the signal constellation and mapping and explaining the reference signals (continual pilots, scattered pilots and TPS) which are needed for the detection algorithms (chapter 2).
- Explanation of channel models and pilot signal detection algorithms. Two channel models are introduced: Rayleigh and Rician. Detection algorithms proposed are described: CP Based Sliding Correlation, Time Domain Pilot Based Sliding Correlation and Time domain Pilot in Cyclic Prefixes Based Sliding Correlation (chapter 3 and 4).
- Detailed explanation of the developed C++ application, describing the generation of DVB-T signals, the detection algorithms and the simulations (chapter 5).
- Comments about the results obtained and conclusions (chapter 6).
- Finally, in the Annex the programmed code is attached.

Chapter 2

The DVB-T European Standard

2.1. General consideration

The DVB-T signal is well known due to the European standard that explains it carefully. In this document ETSI EN 300 744 V1.6.1 (2009-01) standard has been used as a reference, where DVB-T signal is completely defined.

The system is directly compatible with MPEG-2 coded TV signals ISO/IEC 13818. It is defined as the functional block unit that performs the adaptation of the baseband TV signals from the output of the MPEG-2 transport multiplexer to the terrestrial channel characteristics. It operates within the existing VHF and UHF spectrum.

The basic performance is as follows: Compressed video, compressed audio and data streams are multiplexed into MPEG program streams (MPEG-PSs). One or more MPEG-PSs are joined together into an MPEG transport stream (MPEG-TS); this is the basic digital stream which is being transmitted and received by TV sets. To ensure that viewers can receive the programs even in the worst environment, two level hierarchy transmissions are used, so video stream (MPEG-TSs) is encoded into high priority and low priority stream.

The functional block diagram of the transmission scheme is showed in figure 2.1.

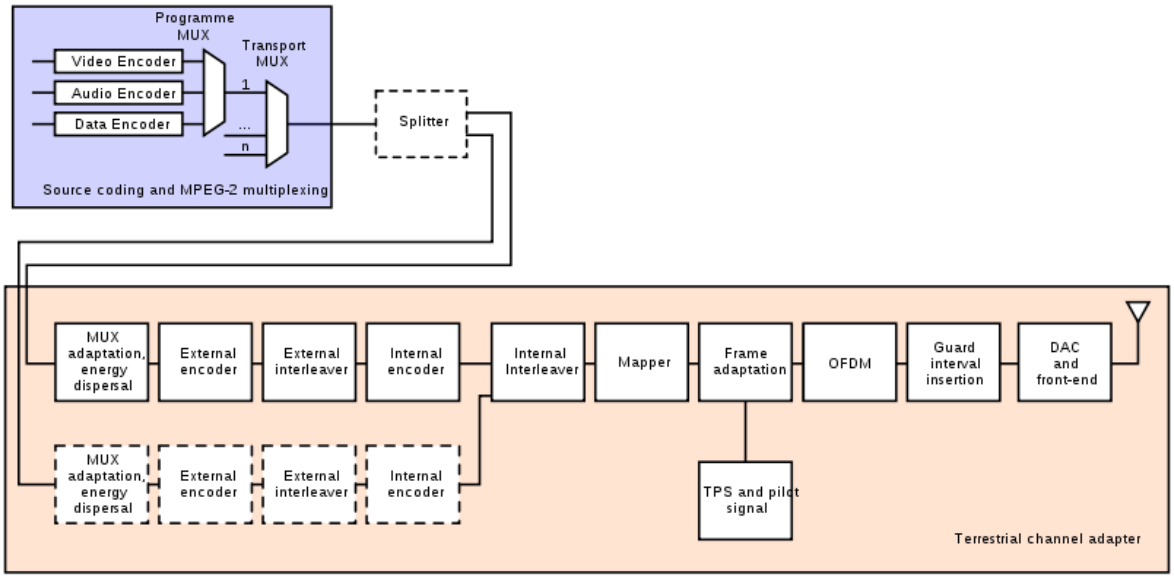

Figure 2.1 DVB-T transmission scheme

The following processes are applied to the data stream:

- **Transport multiplex adaptation and randomization for energy dispersal:** The MPEG-TS is identified as a sequence of data packets of fixed length. The byte sequence is decorrelated and energy dispersal is used to scramble the data bits to prevent energy to concentrate on a single frequency.
- **Outer coding**: A first level of error correction is applied to the transmitted data.
- **Outer interleaving**: Convolutional interleaving is used to rearrange the transmitted data sequence, thus it becomes more rugged to long sequences of errors.
- **Inner coding**: A second level of error correction is given by a punctured convolutional code.
- **Inner interleaving**: A block interleaving technique is adopted with a pseudo-random assignment scheme. Data sequence is rearranged again, aiming to reduce the influence of burst errors.
- **Mapping:** The digital data bit sequence is translated into constellation coordinates of complex symbols.
- **Frame adaptation:** The complex symbols are grouped in blocks of constant length.
- **Pilot and TPS signals:** In order to simplify the reception of the signal being transmitted pilot signals are inserted. Pilot signals are used during the synchronization and equalization phase, while TPS signals send the parameters of the transmitted signal. The receiver must be able to synchronize, equalize, and decode the signal to gain access to the information held by the TPS pilots.
- **OFDM Modulation:** The sequence is modulated according to an OFDM technique.
- **Guard interval insertion:** To decrease receiver complexity, every OFDM block is extended, copying in front of it its own end (cyclic prefix).
- **DAC and front-end:** The digital signal is transformed into an analog signal, with a digital-to-analog converter (DAC), and then modulated to radio frequency (VHF, UHF) by the RF front-end.

In this paper, only mapping, frame adaptation, pilot and TPS signal and OFDM modulation processes are required and are implemented in the C++ application. This is due to the fact that the important information for detecting the signal is contained in the pilot (introduced during the frame adaption), so data used in the simulation is random.

The signal mapping allows different levels of QAM modulation and different inner code rates to be used to trade bit rate versus ruggedness. It allows two level hierarchical channel coding and modulation, including uniform and multi-resolution constellation.

2.2. Specification of DVB-T

The most important characteristics of DVB-T are listed in Table 2.1, and it contains among others the number of sub-carriers, number of pilots, guard interval length, bandwidth and constellations used.

Transmission mode	2k, 8k
Number of useful sub-carriers	1705, 6817
Number of continual pilots	45, 177
Number of scattered pilots	141, 564
Number of TPS pilots	17, 68
Radio frequency (MHz)	45 ~ 860
Guard interval	1/4, 1/8, 1/16, 1/32
Bandwidth (MHz)	6, 7, 8
Elementary period (us)	7/48, 7/56, 7/64
Channel model	Rayleigh, Ricean
Constellation	QPSK, 16QAM, 64QAM, non-uniform 16QAM, non-uniform 16QAM
Required BER	2×10^{-4}

Table 2.1 DVB-T Parameters

Interfacing

The Baseline System is delimited by the following interfaces:

Location	Interface	Interface type	Connection
Transmit Station	Input	MPEG-2 transport stream multiplex	From MPEG-2 multiplexer
	Output	RF signal	To aerial
Receive Installation	Input	RF	From aerial
	Output	MPEG-2 transport stream multiplex	To MPEG-2 demultiplexer

Table 2.2 Interfaces for the Baseline System

2.3. Signal constellation and mapping

Orthogonal Frequency Division Multiplex (OFDM) transmission is used. The signal mapping depends on two parameters, the modulation and α. All data carriers in one OFDM frame are modulated using QPSK, 16-QAM, 64-QAM, non-uniform 16-QAM or non-uniform 64-QAM constellations. The more the QAM number can transmit the more coded data bits (it can provide higher A/V quality). The proportions of the constellations depend on a parameter α which is the minimum distance separating two constellation points. α can take the three values 1, 2 or 4.

The exact values of the constellation points are $z \in \{n + jm\}$ with values of n, m given below:

- **QPSK** (Figure 2.2).

  $n \in$ {-1, 1}, $m \in$ {-1, 1}

- **16-QAM (non-hierarchical and hierarchical with α = 1)** (Figure 2.3).

  $n \in$ {-3, -1, 1, 3}, $m \in$ {-3, -1, 1, 3}

- **Non-uniform 16-QAM with α = 2** (Figure 2.4).

  $n \in$ {-4, -2, 2, 4}, $m \in$ {-4, -2, 2, 4}

- **Non-uniform 16-QAM with α = 4** (Figure 2.5).

  $n \in$ {-6, -4, 4, 6}, $m \in$ {-6, -4, 4, 6}

- **64-QAM (non-hierarchical and hierarchical with α = 1)** (Figure 2.6).

  $n \in$ {-7, -5, -3, -1, 1, 3, 5, 7}, $m \in$ {-7, -5, -3, -1, 1, 3, 5, 7}

- **Non-uniform 64-QAM with α = 2** (Figure 2.7).

  $n \in$ {-8, -6, -4, -2, 2, 4, 6, 8}, $m \in$ {-8, -6, -4, -2, 2, 4, 6, 8}

- **Non-uniform 64-QAM with α = 4** (Figure 2.8).

  $n \in$ {-10, -8, -6, -4, 4, 6, 8, 10}, $m \in$ {-10, -8, -6, -4, 4, 6, 8, 10}

And the representation of the constellation coordinates is shown in 2.2. to 2.8 figures.

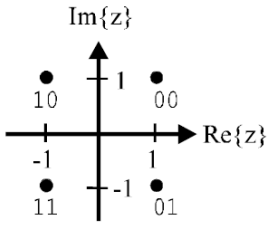

Figure 2.2 QPSK

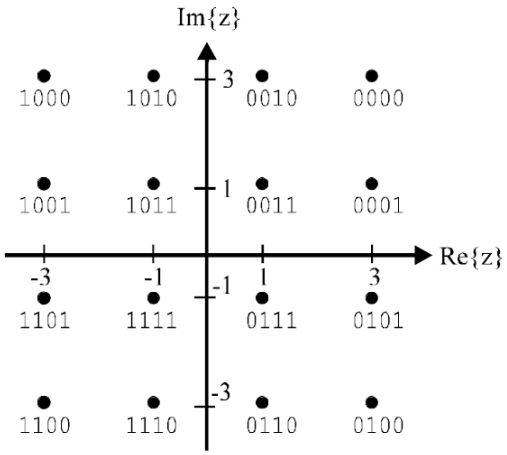

Figure 2.3 Uniform 16 QAM (α = 1)

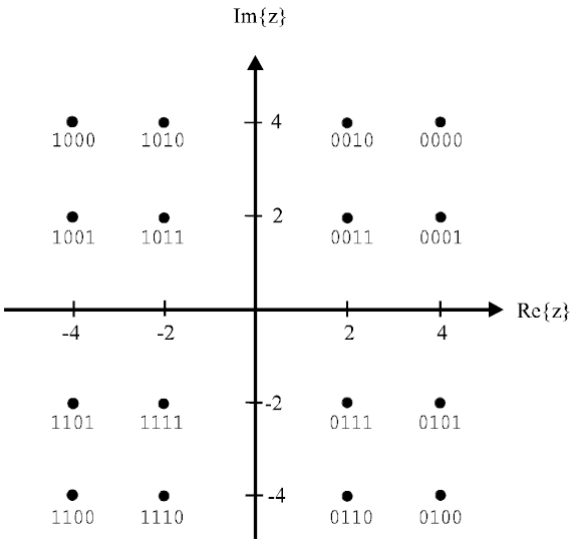

Figure 2.4 Non – uniform 16 QAM (α = 2)

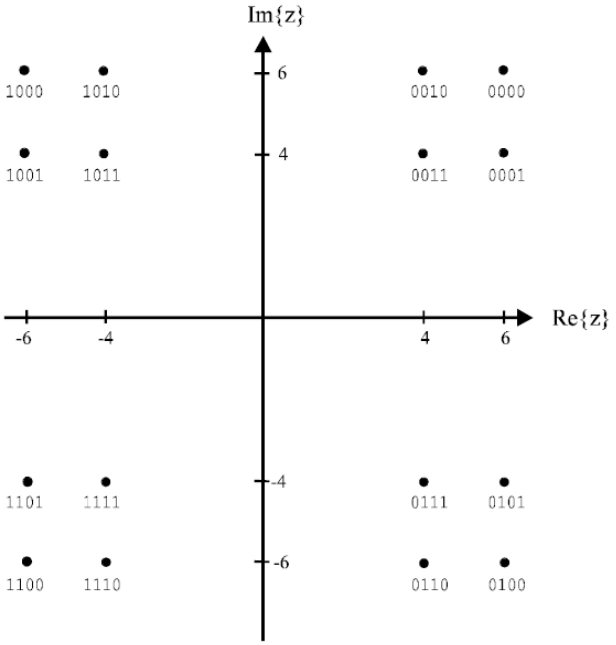

Figure 2.5 Non – uniform 16 QAM (α = 4)

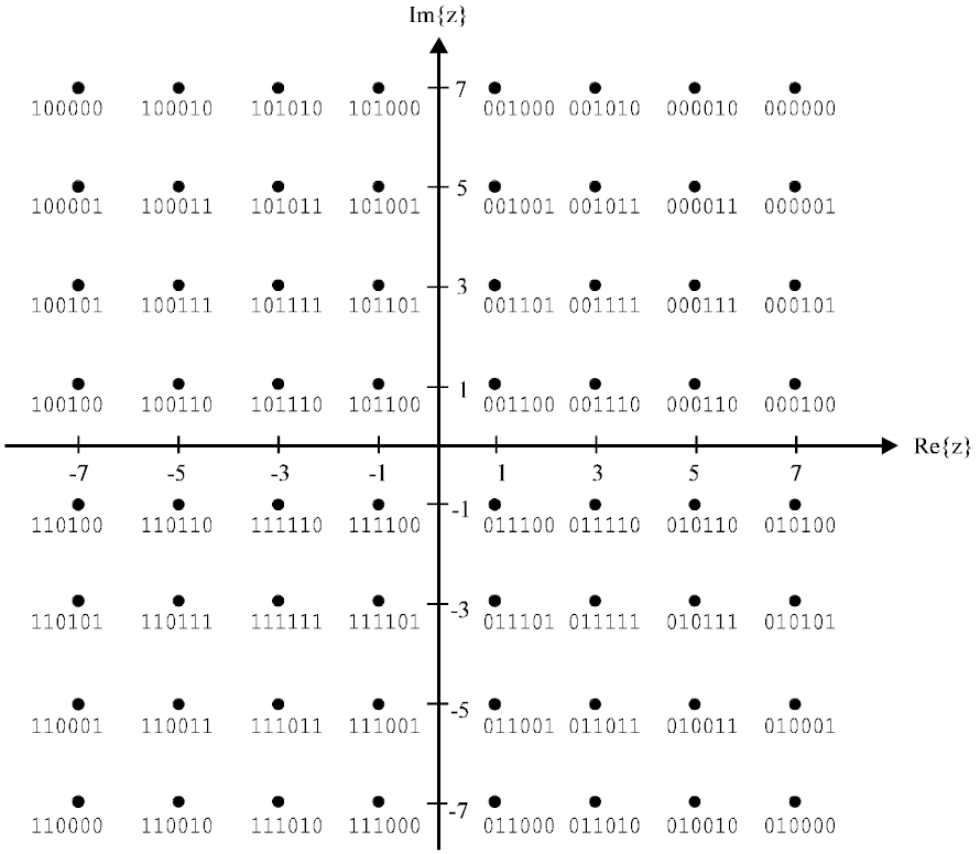

Figure 2.6 Uniform 64 QAM (α = 1)

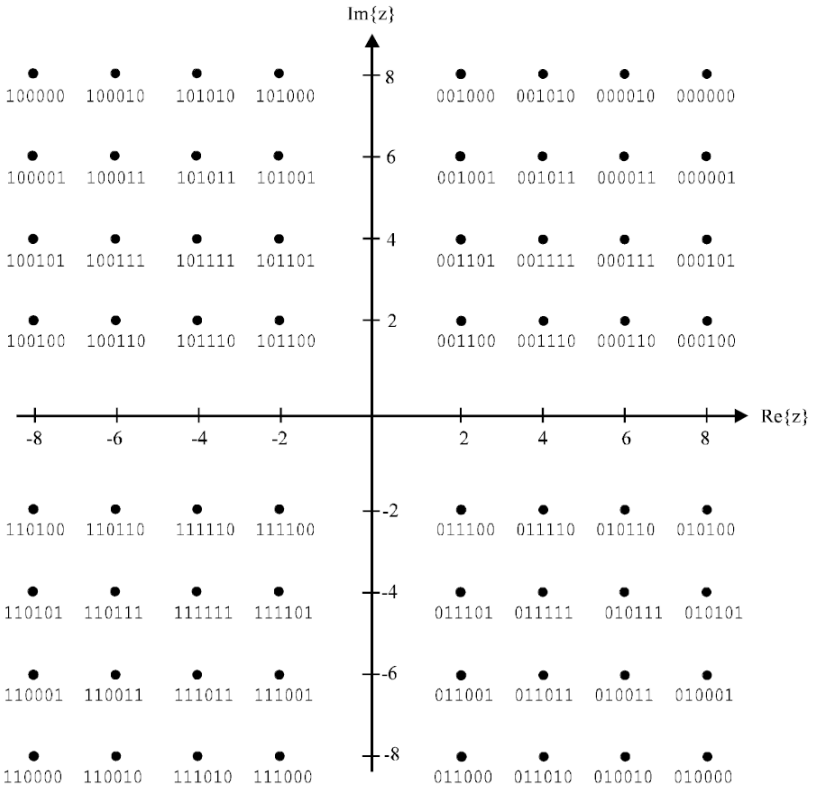

Figure 2.7 Non – uniform 64 QAM (α = 2)

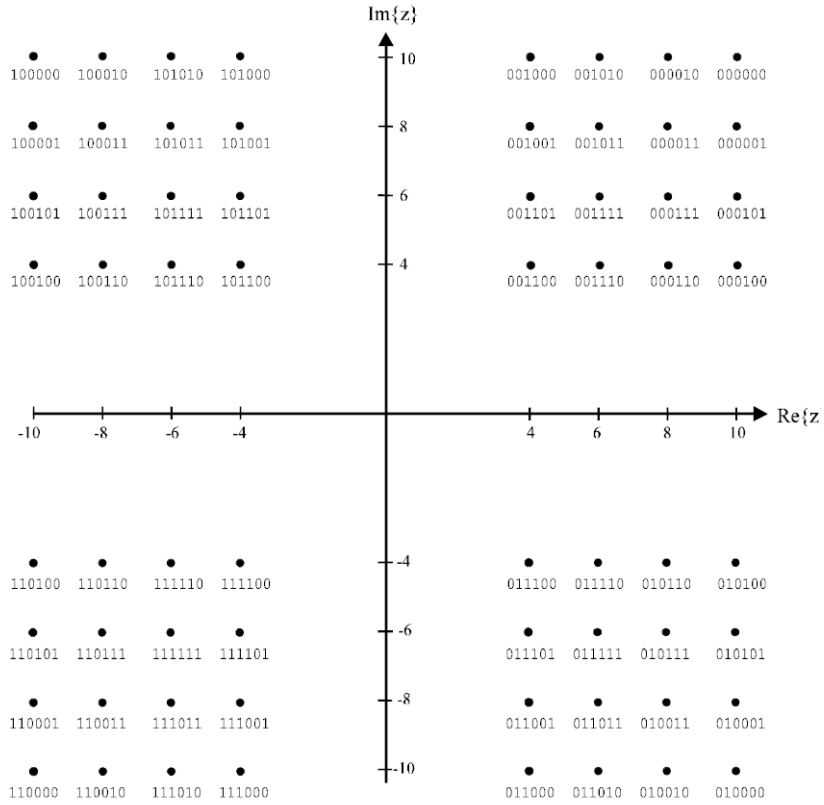

Figure 2.8 Non – uniform 64 QAM (α = 4)

2.4. Concept of OFDM

The basic idea of OFDM is to divide the bandwidth into several sub-channels such that these narrow sub-channels can have flat fading. The feature of orthogonal sub-channels makes OFDM have a high spectral efficiency. Cyclic extension is a copy of the last or the forward part of each OFDM symbol. It prevents inter symbol interference (ISI) and inter carrier interference (ICI), and makes the transmitted signal periodic.

2.4.1. Mathematical Descriptions of OFDM

A continual time model of OFDM symbol can be considered that data $X_k(t)$ are modulated by a series of orthogonal sub-carriers. Assuming it has N_{sc} sub-carriers:

$$S_n(t) = \sum_{k=0}^{N_{sc}} X_k(t) e^{j2\pi k \Delta f t} \quad [2.1]$$

Then, it is sampled by a sampling frequency 1/τ. The duration of one symbol is T_u, which has a relationship as equation 2.2, and N_{sc} samples are generated:

$$T_u = N_{sc}\tau \quad [2.2]$$

If the waveform of data $X_k(t)$ is a fixed value over a symbol period it can be rewritten as $X(k)$. The result can be represented by:

$$S_n(n\tau) = \sum_{k=0}^{N_{sc}} X(k) e^{j2\pi k \Delta f n\tau} \quad [2.3]$$

Comparing the general form of inverse Fourier transform shown in equation 2.4 with equation 2.3, if equation 2.5 then equation 2.2 and equation 2.4 are equivalent. Therefore, IDFT can be used to implement the modulation of an OFDM system:

$$y(n\tau) = \frac{1}{N_{sc}} \sum_{k=0}^{N_{sc}} X(k) e^{j2\pi kn\tau / N_{sc}} \quad [2.4]$$

$$\Delta f = \frac{1}{N_{sc}\tau} \quad [2.5]$$

2.4.2. Guard Interval and Cyclic Extension

ISI and ICI introduced by transmission channel distortion do damage on the orthogonality of sub-carriers in OFDM. A solution is to add an empty guard interval (GI) between two consecutive symbols. If the length of GI is longer than the delay spread of channel response, the next symbol doesn't interfere with the previous one. But if symbol boundary estimation doesn't precisely locate the symbol, the empty GI destroys the orthogonality and introduces ICI. In order to prevent this

situation, a mechanism is proposed to copy the last part of an OFDM symbol into the empty GI, which is so called Cyclic Prefix (CP) as shown in Figure 2.9.

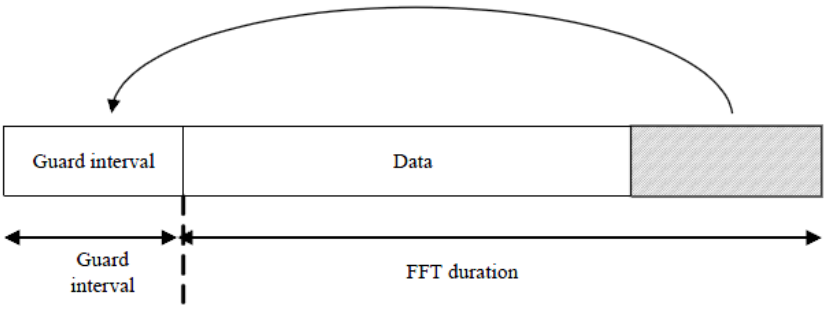

Figure 2.9 Cyclic prefix and Guard interval

CP makes the sub-carrier signal has integral periods, so it can maintain the orthogonality. Besides, adding CP to each OFDM symbol makes line convolution be equivalent to a circular convolution. The l^{th} received OFDM symbol $r_{l,n}$ and its DFT $R_{l,k}$ is described as:

$$\begin{aligned} R_{l,k} &= DFT\{r_{l,n}\} \\ &= DFT\{IDFT\{D_{l,k}\} \otimes h_{l,n}\} \\ &= DFT\left\{IDFT\{D_{l,k}\}\right\} DFT\{h_{l,n}\} \\ &= D_{l,k} \cdot H_{l,k} \end{aligned} \qquad [2.6]$$

Because of the principle of circular convolution, the transmitted data $D_{l,k}$ can be recovered by the estimation of response of channel.

$$\widehat{D}_{l,k} = \frac{R_{l,k}}{\widehat{H}_{l,k}} \qquad [2.7]$$

Where k is the sub-carrier index, n is the sample index in time domain, $h_{l,n}$ is the channel impulse response and $H_{l,k}$ is the channel frequency response.

2.4.3. OFDM frame structure

The OFDM system is specified for 8 MHz, 7 MHz and 6 MHz channel spacing. A flexible guard interval is specified to allow optimal tradeoff between network topology and frequency efficiency. This will enable the system to support different network configurations, such as large area SFN and single transmitter. Two modes of operation are defined, a 2K mode and an 8K mode.

- The "2k mode" has wider sub-carriers spacing, so it can be used against the distortion caused by Doppler spread. It is suitable for single transmitter operation and for small SFN networks with limited transmitter distances.
- The "8k mode" has longer symbol duration, so it is adapted for long distance transmission. It can be used both for single transmitter operation and for small and large SFN networks.

The OFDM transmitted signal is organized in frames. One super-frame is constituted by four frames. Each frame consists of 68 OFDM symbols and has duration of T_F. Each symbol is constituted by a set of K carriers and transmitted with duration T_S. Depending on the mode, $k = 6.817$ carriers (8k mode) or $k = 1.705$ carriers (2k mode).

The symbol is composed of two parts, a useful part with duration T_U and a guard interval with duration Δ. The guard interval consists in a cyclic continuation of the useful part, T_U, and is inserted before it. Four values of guard intervals are used $^1/_4, ^1/_8, ^1/_{16}$ and $^1/_{32}$.

The symbols in an OFDM frame are numbered from 0 to 67. All symbols contain data and reference information. In addition to the transmitted data an OFDM frame contains:

- Continual pilot carriers.
- Scattered pilot cells.
- TPS carriers.

The pilots are used for frame synchronization, frequency synchronization, time synchronization, channel estimation, transmission mode identification and can also be used to follow the phase noise.

The carriers are indexed by $k \in [K_{min}; K_{max}]$ and determined by $K_{min} = 0$ and $K_{max} = 1.704$ in 2K mode and $K_{max} = 6.816$ in 8K mode respectively. The spacing between adjacent carriers is $\frac{1}{T_u}$ while the spacing between carriers K_{min} and K_{max} is determined by $\frac{K-1}{T_u}$.

The values for the various time-related parameters are given in multiples of the elementary period T. The elementary period T can be obtained by:

$$T[\mu s] = \frac{7}{8 \cdot Bw\ [MHz]} \qquad [2.8]$$

So, values of T for the specified bandwidth are $^{7}/_{48}\ \mu s$ for 6 MHz channels, $^{1}/_{8}\ \mu s$ for 7 MHz channels and $^{7}/_{64}\ \mu s$ for 8 MHz channels.

Numerical values for the OFDM parameters for the 8K and 2K modes for 8 MHz channels are shown in table 2.3.

Parameter	8K mode	2K mode
Number of carriers K	6.817	1.705
Value of carrier number K_{min}	0	0
Value of carrier number K_{max}	6.816	1.704
Duration T_u	896 us	224 µs
Carrier spacing $\frac{1}{T_u}$	1.116 Hz	4.464 Hz
Spacing between carriers K_{min} and K_{max} . $\frac{K-1}{T_u}$	7,61 MHz	7,61 MHz

Table 2.3 Numerical values for the OFDM parameters for the 8K and 2K modes for 8 MHz channels

The emitted signal $s(t)$ is described by the following expression:

$$s(t) = Re\left\{e^{j\,2\pi f_c t} \sum_{m=0}^{\infty} \sum_{l=0}^{67} \sum_{k=k_{min}}^{k_{max}} c_{m,l,k} \cdot \varphi_{m,l,k}(t)\right\} \qquad [2.9]$$

$$\varphi_{m,l,k}(t) = \begin{cases} e^{j\,2\pi \frac{k'}{T_u}(t-\Delta-l\,\cdot\,T_s-68\cdot m\cdot T_s)} & (l+68\cdot m)\cdot T_s \leq t \leq (l+68\cdot m+1)\cdot T_s \\ 0 & else \end{cases} \qquad [2.10]$$

Where k is the carrier number, l is the OFDM symbol number, m is the transmission frame number, K is the number of transmitted carriers, T_s is the symbol duration, T_u is the inverse of the carrier spacing, Δ is the duration of the guard interval, f_c is the central frequency of the RF signal, k' is the carrier index relative to the centre frequency, $k' = k - \frac{k_{max}+k_{min}}{2}$, $c_{m,0,k}$ is the complex symbol for carrier k of the Data symbol number 1 in frame number m, $c_{m,1,k}$ is the complex symbol for carrier k of the Data symbol number 2 in frame number m and $c_{m,67,k}$ is the complex symbol for carrier k of the Data symbol number 68 in frame number m.

Different values of the duration of the symbol part, duration of the guard interval, symbol duration are shown on the table 2.4 depending on guard interval.

Mode	8K mode				2K mode			
Guard Interval	$^1/_4$	$^1/_8$	$^1/_{16}$	$^1/_{32}$	$^1/_4$	$^1/_8$	$^1/_{16}$	$^1/_{32}$
Duration of symbol part	$8192 \cdot T$ $896\ \mu s$				$2048 \cdot T$ $224\ \mu s$			
Duration of guard interval	$2048 \cdot T$ $224\ \mu s$	$1024 \cdot T$ $112\ \mu s$	$512 \cdot T$ $56\ \mu s$	$256 \cdot T$ $28\ \mu s$	$512 \cdot T$ $56\ \mu s$	$256 \cdot T$ $28\ \mu s$	$128 \cdot T$ $14\ \mu s$	$64 \cdot T$ $7\ \mu s$
Symbol duration	$10240 \cdot T$ $1120\ \mu s$	$9216 \cdot T$ $1008\ \mu s$	$8704 \cdot T$ $952\ \mu s$	$8448 \cdot T$ $924\ \mu s$	$2560 \cdot T$ $280\ \mu s$	$2304 \cdot T$ $252\ \mu s$	$2176 \cdot T$ $238\ \mu s$	$2112 \cdot T$ $231\ \mu s$

Table 2.4 Duration of symbol part for the allowed guard intervals for 8 MHz channels

The $c_{m,l,k}$ values are normalized modulation values of the constellation z. The normalization factors are that $E[c\ x\ c^*] = 1$. Values of normalization are shown in table 2.5 according to the modulation scheme.

Modulation Scheme	α	Normalization factor
QPSK		$c = \frac{z}{\sqrt{2}}$
16 QAM	$\alpha = 1$	$c = \frac{z}{\sqrt{10}}$
16 QAM	$\alpha = 2$	$c = \frac{z}{\sqrt{20}}$
16 QAM	$\alpha = 4$	$c = \frac{z}{\sqrt{52}}$
64 QAM	$\alpha = 1$	$c = \frac{z}{\sqrt{42}}$
64 QAM	$\alpha = 2$	$c = \frac{z}{\sqrt{60}}$
64 QAM	$\alpha = 4$	$c = \frac{z}{\sqrt{108}}$

Table 2.5 Normalization factors for data symbols

2.5. Reference Signals

2.5.1. Definition of reference sequence.

Various cells within the OFDM frame are modulated with reference information whose transmitted value is known to the receiver. Cells containing reference information are transmitted at "boosted" power level. The information transmitted in these cells is scattered or continual pilot cells.

There are three kinds of pilot signals in DVB-T, these are continual pilot carrier, scattered pilot cells, and Transmission Parameter Signal (TPS) carrier. These pilot signals are used for synchronization, estimating channel response, and transmitting system information.

The continual and scattered pilots are modulated according to a PRBS sequence, w_k, corresponding to their respective carrier index k. This sequence also governs the starting phase of the TPS information. The PRBS is initialized so that the first output bit from the PRBS coincides with the first active carrier. A new value is generated by the PRBS on every used carrier (whether or not it is a pilot).

The polynomial for the Pseudo Random Binary Sequence is $w_k = X^{11} + X^2 + 1$ and is generated according to figure 2.10.

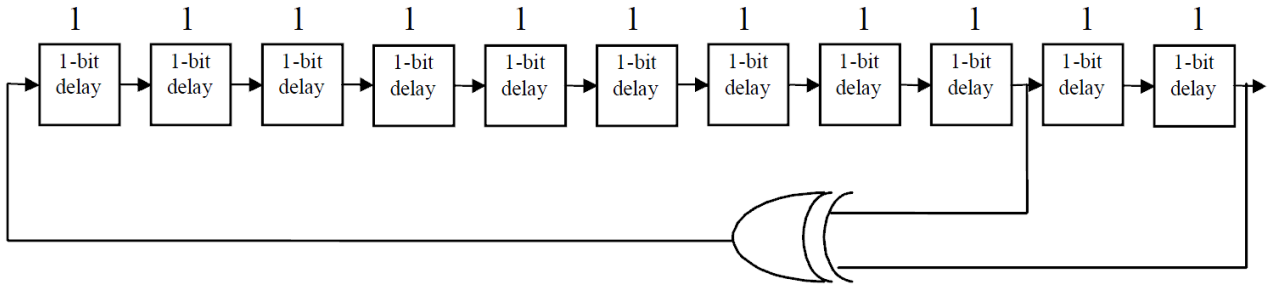

Figure 2.10 Generation of PRBS sequence

2.5.2. Continual pilot carriers

There are 177 continual pilots in the 8k mode and 45 in the 2k mode. Continual Pilots are inserted into fixed sub-carriers at each transmitted symbol (they occur on all symbols).

All continual pilots are modulated according to the reference sequence. Each continual pilot coincides with a scattered pilot every four symbols; the number of useful data carriers is constant from symbol to symbol: 1 512 useful carriers in 2K mode and 6 048 useful carriers in 8K mode

Both continual pilots and scatted pilots are transmitted in boosted powered level 16/9 (square of 4/3), and the power level of data carriers is normalized to 1. The power of continual pilots and scattered pilots higher than other data carriers helps synchronization and channel estimation.
The transmitted value of continual pilots is modulated by:

$$\mathrm{Re}\{c_{m,l,k}\} = \frac{4}{3} \cdot 2 \cdot \left(\frac{1}{2} - w_k\right) \qquad \mathrm{Im}\{c_{m,l,k}\} = 0 \qquad [2.11]$$

Where c is the transmission signal of continual pilots, m is the frame index, l is the OFDM symbol index, k is the index of continual pilot position and w_k is the PRBS sequence.

The position of the continual pilot is listed in Table 2.6.

Continual pilot carrier positions (index number k)	
2k mode	**8k mode**
0 48 54 87 141 156 192 201 255 279 282 333 432 450 483 525 531 618 636 714 759 765 780 804 873 888 918 939 942 969 984 1050 1101 1107 1110 1137 1140 1146 1206 1269 1323 1377 1491 1683 1704	0 48 54 87 141 156 192 201 255 279 282 333 432 450 483 525 531 618 636 714 759 765 780 804 873 888 918 939 942 969 984 1050 1101 1107 1110 1137 1140 1146 1206 1269 1323 1377 1491 1683 1704 1752 1758 1791 1845 1860 1896 1905 1959 1983 1986 2037 2136 2154 2187 2229 2235 2322 2340 2418 2463 2469 2484 2508 2577 2592 2622 2643 2646 2673 2688 2754 2805 2811 2814 2841 2844 2850 2910 2973 3027 3081 3195 3387 3408 3456 3462 3495 3549 3564 3600 3609 3663 3687 3690 3741 3840 3858 3891 3933 3939 4026 4044 4122 4167 4173 4188 4212 4281 4296 4326 4347 4350 4377 4392 4458 4509 4515 4518 4545 4548 4554 4614 4677 4731 4785 4899 5091 5112 5160 5166 5199 5253 5268 5304 5313 5367 5391 5394 5445 5544 5562 5595 5637 5643 5730 5748 5826 5871 5877 5892 5916 5985 6000 6030 6051 6054 6081 6096 6162 6213 6219 6222 6249 6252 6258 6318 6381 6435 6489 6603 6795 6816

Table 2.6 Carrier indices for continual pilot carriers

2.5.3. Scattered pilot cells

There are 564 scattered pilots in the 8k mode and 141 in the 2k mode. All scattered pilots are modulated according to the reference sequence and are always modulated at the boosted power level.

The scattered pilot is used to estimate channel response, because the channel response of data carriers can be estimated by interpolating the stored scattered pilots. The periodic change of position gets better performance of channel estimation than the continual pilots who have fix position.

As continual pilots, the transmitted value of scattered pilots is modulated by:

$$\mathrm{Re}\{c_{m,l,k}\} = \frac{4}{3} \cdot 2 \cdot (\frac{1}{2} - w_k) \qquad \mathrm{Im}\{c_{m,l,k}\} = 0 \qquad [2.12]$$

Where c is the transmission signal of scattered pilots, m is the frame index, l is the OFDM symbol index, k is the index of continual pilot position and w_k is the PRBS sequence.

The location of scattered pilots is periodic in every four OFDM symbols and the detail is specified by Equation (2.12):

$$k = K_{min} + 3 \cdot (l \bmod 4) + 12p \qquad p \in int, p \geq 0 \quad k \in [K_{min} : K_{max}] \qquad [2.13]$$

Where l is the OFDM symbol index, k is the position of scattered pilots, $K_{min} = 0$ for all mode, $K_{max} = 6.816$ for 8k mode and $K_{max} = 1.704$ for 2k mode.

The pilot insertion patter is shown in figure 2.11.

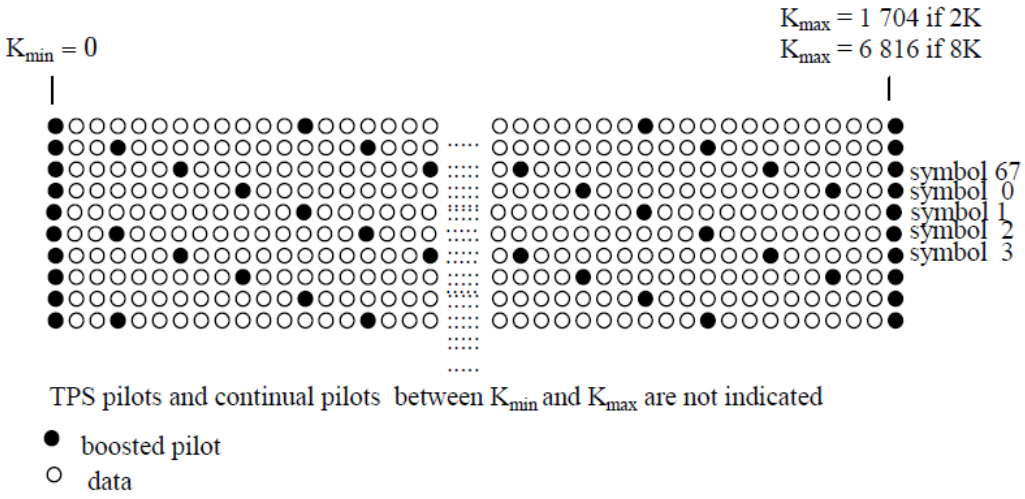

Figure 2.11 Distribution of scattered pilots

2.5.4. Transmission parameter signaling (TPS)

DVB-T is a broadcasting transmission system, so it has no handshaking before communication. DVB-T inserts the transmission parameters in some certain pilots which are called TPS pilot.

The TPS carriers are used for the purpose of signalling parameters related to the transmission scheme. The TPS is defined over 68 consecutive OFDM symbols and transmitted in parallel on 17 TPS carriers for the 2K mode and on 68 TPS carriers for the 8K mode. Every TPS carrier in the same

symbol conveys the same differentially encoded information bit. The following carrier indices contain TPS carriers:

TPS carrier positions (index number k)	
2k mode	**8k mode**
34 50 209 346 413 569 595 688 790 901 1073 1219 1262 1286 1469 1594 1687	34 50 209 346 413 569 595 688 790 901 1073 1219 1262 1286 1469 1594 1687 1738 1754 1913 2050 2117 2273 2299 2392 2494 2605 2777 2923 2966 2990 3173 3298 3391 3442 3458 3617 3754 3821 3977 4003 4096 4198 4309 4481 4627 4670 4694 4877 5002 5095 5146 5162 5321 5458 5525 5681 5707 5800 5902 6013 6185 6331 6374 6398 6581 6706 6799

Table 2.7 Carrier indices for TPS carriers

The TPS carriers contain information about:

- Modulation including α value of the QAM constellation pattern.
- Hierarchy information.
- Guard interval.
- Inner code rates.
- Transmission mode.
- Frame number in a super-frame.
- Cell identification.

2.5.4.1. Definition of the TPS

The TPS is defined over 68 consecutive OFDM symbols, referred to as one OFDM frame. The reference sequence corresponding to the TPS carriers of the first symbol of each OFDM frame are used to initialize the TPS modulation on each TPS carrier.

Each OFDM symbol conveys one TPS bit. Each TPS block (corresponding to one OFDM frame) contains 68 bits, defined as follows:

- 1 initialization bit.
- 16 synchronization bits.

- 37 information bits.
- 14 redundancy bits for error protection.

Of the 37 information bits, 31 are used at present. The remaining 6 bits are reserved for future use, and should be set to zero.

2.5.4.2. TPS modulation

Every TPS carrier is DBPSK modulated and conveys the same message. The DBPSK is initialized at the beginning of each TPS block. TPS cells are transmitted at the "normal" power level, (they are transmitted with energy equal to that of the mean of all data cells $E[c\ x\ c^*] = 1$.

The initial modulation of the TPS carriers in the first symbol in a frame is derived from the reference sequence w_k as follows:

$$Re\{c_{m,l,k}\} = 2 \cdot ({}^{1}/_{2} - w_k); \qquad Im\{c_{m,l-1,k}\} = 0; \qquad [2.14]$$

Where c is the transmission signal of TPS pilots, m is the frame index, l is the OFDM symbol index and k is the index of TPS position.

The following rule applies for the differential modulation of carrier k of symbol l ($l > 0$) in frame m:

- $if\ s_l = 0, then\ Re\{c_{m,l,k}\} = Re\{c_{m,l-1,k}\}; \qquad Im\{c_{m,l-1,k}\} = 0; \qquad [2.15]$
- $if\ s_l = 1, then\ Re\{c_{m,l,k}\} = -Re\{c_{m,l-1,k}\}; \qquad Im\{c_{m,l-1,k}\} = 0; \qquad [2.16]$

Where s_l is the TPS at l^{th} OFDM symbol.

2.5.4.3. TPS transmission format

The transmission parameter information is transmitted as shown in table 2.8.

Bit number	Purpose/Content
0	Initialization
1 – 16	Synchronization word
17 – 22	Length indicator
23 – 24	Frame number

25 – 26	Constellation
27 – 29	Hierarchy information
30 – 32	Code rate, HP stream
33 – 35	Code rate, LP stream
36 – 37	Guard interval
38 – 39	Transmission mode
40 – 53	Reserved for future use
54 – 67	Error protection

Table 2.8 TPS signalling information and format

2.5.4.3.1. Initialization

The first bit s_0 is an initialization bit for the differential 2-PSK modulation. The modulation of the TPS initialization bit is derived from the PRBS sequence.

2.5.4.3.2. Synchronization

Bits s_1 to s_{16} of the TPS are the synchronization word. TPS blocks have different synchronization word, depending if the super-frame is odd or even.

The first and third TPS block in each super-frame have the following synchronization word:

$$s_1 - s_{16} = 0011010111101110$$

The second and fourth TPS block have the following synchronization word:

$$s_1 - s_{16} = 1100101000010001$$

2.5.4.3.3. TPS length indicator

The first 6 bits of the TPS information is used as a TPS length indicator (binary count) to signal the number of used bits of the TPS. Depending if the cell identification is or not supported. The value of $s_{17} - s_{22}$ are:

- Supported: $s_{17} - s_{22} = 0111111$.
- Not supported: $s_{17} - s_{22} = 0100111$.

2.5.4.3.4. Frame number

One super-frame is constituted by four frames. The frames inside the super-frame are numbered from 0 to 3 according to table 2.9.

Bits s_{23}, s_{24}	Frame number in the super - frame
00	1
01	2
10	3
11	4

Table 2.9 Signalling format for frame number

2.5.4.3.5. Constellation

The constellation is signalled by 2 bits according to table 2.10.

Bits s_{25}, s_{26}	Constellation characteristics
00	QPSK
01	16 – QAM
10	64 – QAM
11	Reserved

Table 2.10 Signalling format for the possible constellation patterns

2.5.4.3.6. Hierarchy information

The hierarchy information specifies whether the transmission is hierarchical and, if so, what the α value is. Where α is signalled by three bits according to table 2.11.

Bits s_{27}, s_{28}, s_{29}	α value
000	Non hierarchical
001	$\alpha = 1$
010	$\alpha = 2$

011	$\alpha = 4$
100	Reserved
101	Reserved
110	Reserved
111	Reserved

Table 2.11 Signalling format for the α value

2.5.4.3.7. Code rates

Non-hierarchical channel coding and modulation requires signalling of one code rate r. In this case, three bits specifying the code rate are followed by another three bits of value 000. Two different code rates may be applied to two different levels of the modulation with the aim of achieving hierarchy. Transmission then starts with the code rate for the HP level (r1) of the modulation and ends with the one for the LP level (r2). Each code rate is signalled according to table 2.12.

Bits s_{30}, s_{31}, s_{32} (HP stream) Bits s_{33}, s_{34}, s_{35} (LP stream)	Code rate
000	$^1/_2$
001	$^2/_3$
010	$^3/_4$
011	$^5/_6$
100	$^7/_8$
101	reserved
110	reserved
111	reserved

Table 2.12 Signalling format for each of the code rates

2.5.4.3.8. Guard intervals

The value of the guard interval is signalled according to table 2.13.

Bits s_{36}, s_{37}	Guard interval values (Δ/T_u)
00	$1/32$
01	$1/16$
10	$1/8$
11	$1/4$

Table 2.13 Signalling format for each of the guard interval values

2.5.4.3.9. Transmission mode

Two bits are used to signal the transmission mode (2K mode or 8K mode). Each mode is signalled according to table 2.14.

Bits s_{38}, s_{39}	Transmission mode
00	2K mode
01	8K mode
10	Reserved
11	Reserved

Table 2.14 Signalling format for transmission mode

2.5.4.3.10. Error protection of TPS

The 53 bits containing the TPS synchronization and information (bits $s_1 - s_{53}$ are extended with 14 parity bits of the $BCH\ (67, 53, t = 2)$ shortened code, derived from the original systematic $BCH\ (127, 113, t = 2)$ code.

Code generator polynomial is:

$$h(x) = x^{14} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1 \qquad [2.17]$$

The shortened BCH code may be implemented by adding 60 bits, all set to zero, before the information bits input of an $BCH\ (127, 113, t = 2)$ encoder. After the BCH encoding these null bits are discarded, leading to a BCH code word of 67 bits.

Chapter 3

Channel Model

Two types of channel are specified in DVB-T standard: Rician (a model for fixed reception) and Rayleigh (a model for portable reception).

The multi-paths fading is modeled by three parameters, power delay, time delay and phase rotation as it is shown in table 3.1.

l	ρ_i	τ_i [μs]	θ_i [rad]
1	0,057662	1,003019	4,855121
2	0,176809	5,422091	3,419109
3	0,407163	0,518650	5,864470
4	0,303585	2,751772	2,215894
5	0,258782	0,602895	3,758058
6	0,061831	1,016585	5,430202
7	0,150340	0,143556	3,952093
8	0,051534	0,153832	1,093586
9	0,185074	3,324866	5,775198
10	0,400967	1,935570	0,154459
11	0,295723	0,429948	5,928383
12	0,350825	3,228872	3,053023
13	0,262909	0,848831	0,628578
14	0,225894	0,073883	2,128544
15	0,170996	0,203952	1,099463
16	0,149723	0,194207	3,462951
17	0,240140	0,924450	3,664773
18	0,116587	1,381320	2,833799
19	0,221155	0,640512	3,334290
20	0,259730	1,368671	0,393889

Table 3.1: Relative power, phase and delay values for F1

Ricean channel is a transmission channel that may have a line-of-sight component and several scattered of multipath components. Rayleigh channel is a communications channel that varies randomly having a fading envelope in the form of the Rayleigh probability density function.

The major difference between Rayleigh and Rician channel is the main path. Rician fading is most applicable when there is a dominant line of sight, if there is no dominant propagation along a line of sight between the transmitter and receiver Rayleigh fading may be more applicable.

In a Rayleigh channel, there is no main path, instead of this, the received signal is reflected into several small power signals, so it is difficult to synchronize.

The Rician factor K, which is the ratio of the power of the direct path (the line of sight ray) to the reflected paths is given by:

$$\mathrm{K} = \frac{\rho_0{}^2}{\sum_{i=1}^{N} \rho_i{}^2} \qquad [3.1]$$

The Rician factor used in the simulations is K = 10 dB. The, the attenuation for the direct path can be obtained directly by:

$$\rho_0 = \sqrt{10 \sum_{i=1}^{N} \rho_i{}^2} \qquad [3.2]$$

The channel model has been generated from the following equations where x(t) and y(t) are input and output signals:

- Fix Reception Rician:

$$\mathrm{y(t)} = \frac{\rho_0 x(t) + \sum_{i=1}^{N} \rho_i e^{-j\theta_i} x(t-\tau_i)}{\sqrt{\sum_{i=1}^{N} \rho_i{}^2}} \qquad [3.3]$$

- Portable reception Rayleigh

$$\mathrm{y(t)} = \frac{\sum_{i=1}^{N} \rho_i e^{-j\theta_i} x(t-\tau_i)}{\sqrt{\sum_{i=1}^{N} \rho_i{}^2}} \qquad [3.4]$$

Where:

- N is the number of echoes equals to 20.
- θ_i is the phase shift from scattering of the i^{th} path.
- ρ_i is the attenuation of the i^{th} path.
- τ_i is the relative delay of the i^{th} path.
- $\rho_0 x(t)$ represents the line of sight ray.

Obviously, the Rician channel has a strong main path, so it is easier to do on synchronization

Finally an additive white Gaussian noise (AWGN) is added to the signal y(t) obtained. This noise is modeled as independent complex Gaussian random variables.

Chapter 4

Detection of DVB-T Signals

The aim of this part is to introduce some sensing algorithms to detect the emitted DVB-T signal. All these algorithms are used for a single antenna receptor.

The primary goal of spectrum sensing is to determine whether a TV channel is occupied by a primary user or is vacant. Of course, identification of which TV channels are occupied and which are unoccupied is complicated by many factors, such as, noise in the receiver, shadow fading, multipath fading, interference arising from wireless transmissions.

The problem of determining if there is a DVB-T signal on a specific spectrum band can be categorized into the following binary hypothesis: the null hypothesis H_0 corresponding to DVB-T signal absent and the alternative hypothesis H_1 corresponding to DVB-T signal existing. These two hypotheses can be represented with the expressions below:

$$H_0: \quad r(k) = n(k) \qquad [4.1]$$

$$H_1: \quad r(k) = \sum_{l=0}^{L_k-1} h_l s(k-l) + n(k) \qquad [4.2]$$

where $r(k)$ represents the k^{th} sample of received signal, $s(k)$ and $h_l (l = 0, \dots, L_k - 1)$ are transmitted signal and the l^{th} path fading coefficient of the channel respectively. L_k is the number of paths and $n(k)$ is the additive white Gaussian noise (AWGN).

There are some distinct characteristics of DVB-T signal such as Cyclic Prefix (CP) or Pilot symbols that make it very interesting to detect the signal in an efficient way. So based on this detection model and the characteristics of DVB-T signal described in chapter 2, several sensing algorithms are proposed:

- Energy Detection.
- Autocorrelation Coeficient Detection.
- Cyclic Prefix based sliding correlation.
- Time domain pilot based sliding correlation (IFFT of the pilots)
- Time domain pilot in Cyclic Prefixes based sliding correlation (IFFT of the pilots).

4.1. Energy Detection (Power detector)

The energy detector is the simplest detector that can be constructed in practice. This detector uses very limited a priori information regarding the DVB-T signal. The detection is based only on the signal power.

Though this detector is unlikely to be used in practice it does give a lower bound on sensing performance, since it is likely that other complex detectors will outperform the power detector. The performance of more complex detectors should be measured against the power detector to evaluate if the higher complexity and sophisticated techniques results in significant performance gains.

The power detector and the energy detector are very similar. The test statistic for the power detector is an estimate of the signal power, while, the test statistic for the energy detector is an estimate of the signal energy during the sensing time.

The test statistic is an estimate of the received signal power which is given by:

$$\Lambda_{ED} = \frac{1}{T_s}\sum_{k=1}^{N} r(k)r^*(n) \qquad [4.3]$$

T_s represents the sensing time which is defined as the product of number of samples with the sampling rate. This is represented as:

$$T_s = \frac{N}{f_s} \qquad [4.4]$$

We sample at the bandwidth Bw so that $f_s = Bw$. The test statistic is then

$$\Lambda_{ED} = \frac{Bw}{N}\sum_{k=1}^{N} r(k)r^*(k) \qquad [4.5]$$

4.2. Autocorrelation Coefficient Detection

The autoccorelation coefficient detector is a simple and computationally efficient spectrum sensing scheme for Orthogonal Frequency Division Multiplexing (OFDM) based primary user signal using its autocorrelation Coefficient.

An OFDM signal consists of a sum of narrowband subcarriers that are typically modulated by using phase shift keying (PSK) or quadrature amplitude keying (QAM). Therefore, T_D represents a number of subcarriers for the OFDM. T_C is the number of symbols in the CP. $x(t)$ is the received complex OFDM signal which is compound as:

$$x(t) = x_r(t) + x_i(t) \quad [4.6]$$

Where $x_r(t)$ and $x_i(t)$ are the real and the imaginary parts of $x(t)$.

For a CP-OFDM signal, the values of the autocorrelation coefficient is $\rho(\tau) = \frac{E[x(t)x^*(t+\tau)]}{E[x(t)x^*(t)]}$ for lags $\tau = \pm T_D$.

Our aim is to devise a detection scheme based on the property above. The observations over several OFDM symbols $x(0), \dots, x(M + T_D - 1)$ where $M \gg T_D$ are used.

The detection statistics can be written as:

$$\Lambda_{AC} = \frac{1}{2(\mathrm{M}+T_D)} \sum_{t=0}^{M+T_D-1} {x_r}^2(t) + {x_i}^2(t) = \quad [4.7]$$

$$\Lambda_{AC} = \frac{1}{2(\mathrm{M}+T_D)} \sum_{t=0}^{M+T_D-1} x^2(t) \quad [4.8]$$

4.3. Cyclic Prefix Based Sliding Correlation

As it has been explained in chapter 2, DVB-T symbol is composed of two parts, a useful part (information data) with duration T_U and a guard interval (Cyclic prefix) with a duration Δ, that consist in a cyclic continuation of the useful part. This guard interval is a signal parameter, so the length of this cyclic prefix is not always the same.

Assuming that there are N carriers in a OFDM symbol and the length of CP is L, then, in time domain, the length of a OFDM symbol is $N + L$. As shown in figure 4.1. the first L samples are the same to the last L samples in the same OFDM symbol. Therefore, a correlation exists between these two parts.

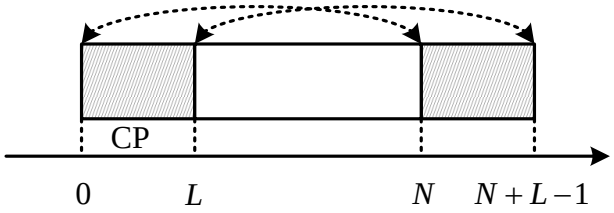

Figure 4.1 OFDM Structure

The detection statistics for this DVB-T signal can be written as:

$$\Lambda_{CP} = \left| \sum_{k=\theta}^{\theta+L-1} r^*(k) r(k+N) \right| \quad [4.9]$$

where θ ranges from o to $N + L - 1$.

This statistic obtains the correlation between all the samples in the cyclic prefix and the L last samples of the OFDM symbol.

The corresponding implementation diagram for this scheme is illustrated in figure 4.2.

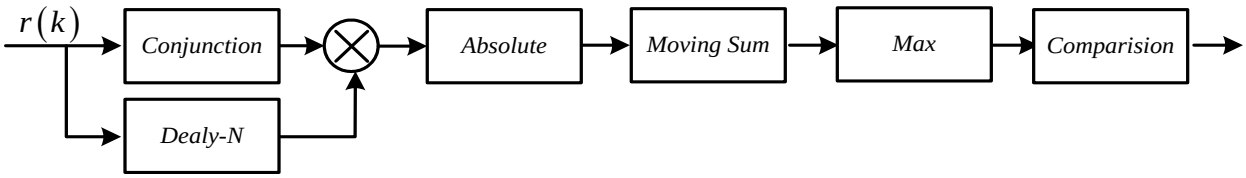

Figure 4.2 Sliding Correlation Based on CP

4.4. Detection Based on IFFT of Pilot

As described in section 2.5, there exist reference signals called pilots (continuous and scattered). This significant characteristic can be utilized for its existence detection.

Assuming that there are N carriers in an OFDM symbol, in the frequency domain and N_p represent the number of the pilot subcarriers. The transmitted DVB-T Signal can be divided into two parts, one with the data carriers and the other one with pilot subcarriers. The first part is made up of $N - N_p$ data carriers and it can be expressed in time domain as:

$$s_t(k) = \frac{1}{\sqrt{N}} \sum_{n \in \{0,\dots,N-1\} \backslash \gamma} x_n e^{j 2\pi k n / N} \qquad [4.10]$$

The second part of the signal is made up of N_p pilot subcarrier and the corresponding representation in time domain is given as:

$$\mathrm{m(k)} = \frac{1}{\sqrt{\mathrm{N}}} \sum_{\mathrm{n}} \mathrm{p_n} \mathrm{e}^{\mathrm{j}2\pi \mathrm{kn}/\mathrm{N}} \qquad [4.11]$$

Therefore the binary hypothesis test can be transformed into the following equations:

$$\mathrm{H_0}: \quad \mathrm{r(k)} = \mathrm{n(k)} \qquad [4.12]$$

$$\mathrm{H_1}: \quad \mathrm{r(k)} = \sum_{\mathrm{l}=0}^{\mathrm{L_k}-1} \mathrm{h_l}[\mathrm{s_l(k-l)} + \mathrm{m(k-l)}] + \mathrm{n(k)} \qquad [4.13]$$

4.4.1. Time Domain Pilot Based Sliding Correlation

In this case, the statistic used here is:

$$\Lambda_{P1} = \left| \sum_{k=\theta}^{\theta+N+L-1} r^*(k) m(k-\theta) \right| \qquad [4.14]$$

Where θ is the time delay between the transmitter and the receiver, $r^*(k)$ is the conjunction of receiving signal $r(k)$. In this algorithm, the value of θ is unknown, so the sliding correlation between r(k) and m(k) is necessary. The statistic obtained with this algorithm is similar as the cyclic prefix based sliding correlation but in this case, the second term only contains the pilots.

The implementation diagram for this scheme is illustrated in figure 4.3.

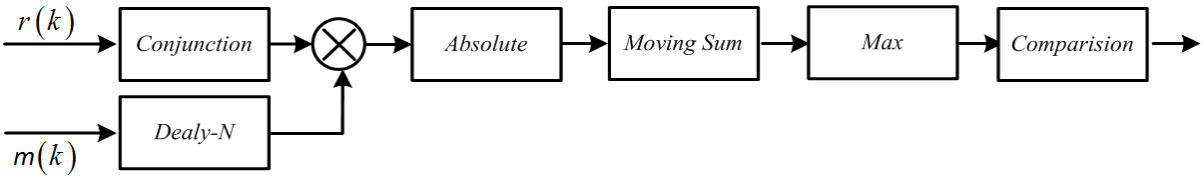

Figure 4.3 Time Domain Pilot Based Sliding Correlation

4.4.2. Time domain Pilot in Cyclic Prefixes Based Sliding Correlation

The length of the time domain pilot based sliding correlation is $N + L - 1$, as can be appreciated in 4.15. That is a very long length to operate and increase the complexity of the whole detection process. In order to simplify the detection process it is needed to reduce the length of this operation. The resulting algorithm uses the two methods described before to get a more efficient algorithm. The statistic used here is:

$$\Lambda_{C2} = \left|\sum_{k=\theta}^{\theta+L-1}(r(k) + r(k+N))^* m(k-\theta)\right| \qquad [4.15]$$

Where $(r(k) + r(k+N))^*$ is the conjunction of $r(k) + r(k+N)$ and θ is the time delay between the transmitter and the receiver.

The correlation studied in this case is between the CP values and the pilots in this CP, so the total length is reduced to L-1.

The implementation diagram of this scheme is depicted in figure 4.4:

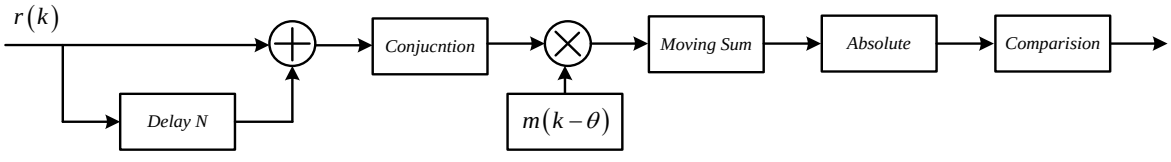

Figure 4.4 Sliding Correlation Based on time domain of Pilot in CP

4.5. Threshold.

To compare the different sensing algorithms, a threshold value is used. This threshold value is calculated based on the detector diagram. The procedure to obtain this value consists on introducing only noise in the detector. After running it several times for different values of random noise, a vector of maximum values is obtained. Then the maximum values are sorted in an increasing order vector and depending on the probability of false alarm desired (P_{fa}), the threshold is calculated. The formula used to calculate the value of the threshold is:

$$Threshold = Max_{value} \cdot \left(1 - P_{fa}\right) \quad [4.16]$$

So, the value of the threshold will be in the in the $\left(\left(1 - P_{fa}\right)\%\right.$ position of the vector.

The probability of false alarm is the probability of detecting an emitted signal when the received signal is only noise. For bigger probabilities of false alarm the threshold will be smaller, so more noise is detected as a signal. Common values for the probability of false alarm are 0.1, 0.01.

Once the threshold value is obtained, the statistic for all the samples is calculated. A maximum value is reached and this value is compared to the threshold to see whether the received signal contains a DVB-T signal or not. If the signal exists (H1) the maximum value will be bigger than the threshold and if there is only noise in the receptor (H0) it will be smaller. This fact is due to the correlation statistic being much bigger in the signal than in the noise.

In the simulations, it will be shown how the probabilities of detecting changes with the probability of false alarm.

Chapter 5

The C++ application

5.1. Parameters.

The generation, detection and simulation of the DVB-T signal have been programmed in a C++ application. For this, I made use of the itpp libraries, which can be found at the following link: *"http://sourceforge.net/apps/wordpress/itpp/"*.

One super frame, that consists in 4 frames of 68 symbols; has been generated as the elementary unit, although the number of super-frames can easily be changed in the simulation. The most important input parameters to model the behavior of the application are showed and explained below:

Generation signals parameters:

- **T**: Sampling interval in seconds.
- **TU**: Useful part duration.
- **Delta**: Guard interval duration.
- **TS**: Total symbol duration.
- **Kmin**: First carrier number.
- **Kmax**: Last carrier number.
- **K**: Number of carriers per symbol.
- **N**: Number of samples per symbol.

Fading parameters:

- **Rho**: Power delay.
- **Taul**: Time delay.
- **Theta:** Phase rotation.

Modulation parameters:

- **BW** (*bandwidth*): The signal is specified for 3 different bandwidth: 6, 7 and 8 MHz's.
- **Mode** (*mode of operation*): Two modes of operation are defined 2 (2K) and 8 (8K).
- **GI** (*guard interval*): Defines the duration of the cyclic continuation of the useful part in each symbol. It is calculated as 2^(-GI). Four values of guard intervals are used: 2, 3, 4 or 5 corresponding to $^1/_4, ^1/_8, ^1/_{16}$ *and* $^1/_{32}$.

- **Level** (*QAM constellation*): The different QAM modulation used in the data carriers. Three values are used 2 for QPSK (2 bits), 4 for 16 QAM (4 bits), 6 for 64 QAM (6 bits).
- **Alpha**: Minimum distance separating two constellation points. Four values are used, 0 for non-hierarchical and 1, 2 and 4 for hierarchical.
- **Cr1, Cr2** (*code rate*): Inner code rates used to trade bit versus ruggedness. Two different code rates may be applied to two different levels of the modulation with the aim of achieving hierarchy (cr1 & cr2). Five values of code rates are defined: $^1/_2, ^2/_3, ^3/_4, ^5/_6$ and $^7/_8$. Each of the parallel channel encoders can have its own code rate. If only one level of hierarchy is used, the second code rate will be set to zero.
- **RF** (*frequency*): Central frequency of the RF signal.

Simulation parameters:

- **T**: The period of time generated in the simulation.
- **Pfa**: Probability of false alarm.
- **Nrof_sims**: Number of simulations.
- **SNR_dB**: Values of the SNR used to plot in the simulation.
- **Threshold:** Two different thresholds have been programmed: theoretical and simulate.
- **Channel:** Three different kinds of channels have been defined: Rayleigh. Rician and AGWN.

5.2. Functions.

For the development of the C++ application the following functions have been defined. A brief explanation of them is shown below:

- **Fading_Parameters**: Model the behavior of the channel. This channel can be Rayleigh, Rician or AWGN.
- **DVBT_Parameters**: Calculate the basic parameters for the generation of the DVB-T signal. These parameters are: BW, mode, GI, TU, delta, TS, Kmin, Kmax, K and N.
- **QAM**: Generate the constellation coordinates for the data.
- **wk_gen**: Generate the PRBS sequence modulated for the scattered and continual pilots.
- **ml_values** (ivec): Generate a vector with the values of l (symbol) and l (frame) for a given “t”.
- **ml_values** (imat): Generate a matrix with the values of l (symbol) and l (frame) and t for a given vector of “t”
- **cmlk_pilots**: Generate a matrix with the continual and scattered pilots.
- **cmlk_tps**: Generate a matrix with the TPS information.

- **cmlk**: Generate a matrix of with all the pilots added (continual, scattered and TPS) and modulated random data.
- **St**: Generate the st function for a number of t samples given.
- **ts**: Calculate a vector of "t" samples, with a separation between them of "TS" seconds.
- **St0**: Generate a vector with the ST signal for the direct path for a time instant (not the matrix, only for K carriers).
- **SG_DVBT:** Generate the complete DVB-T signal.
- **CP_detector**: Model the "Cyclic Prefix Based Sliding Correlation" algorithm.
- **Pilot_detector**: Model the "Time domain Pilot based sliding correlation" algorithm.
- **CP_pilot_detector**: Model the "Time domain Pilot in cyclic prefixes based sliding correlation" algorithm.

5.3. Performance explanation.

With all the parameters and functions described, it is possible to give a complete explanation on the performance of the entire system. The function that generates the signal is St and its structure used to generate this signal using the functions defined in the previous paragraph is illustrated in Figure 5.1.

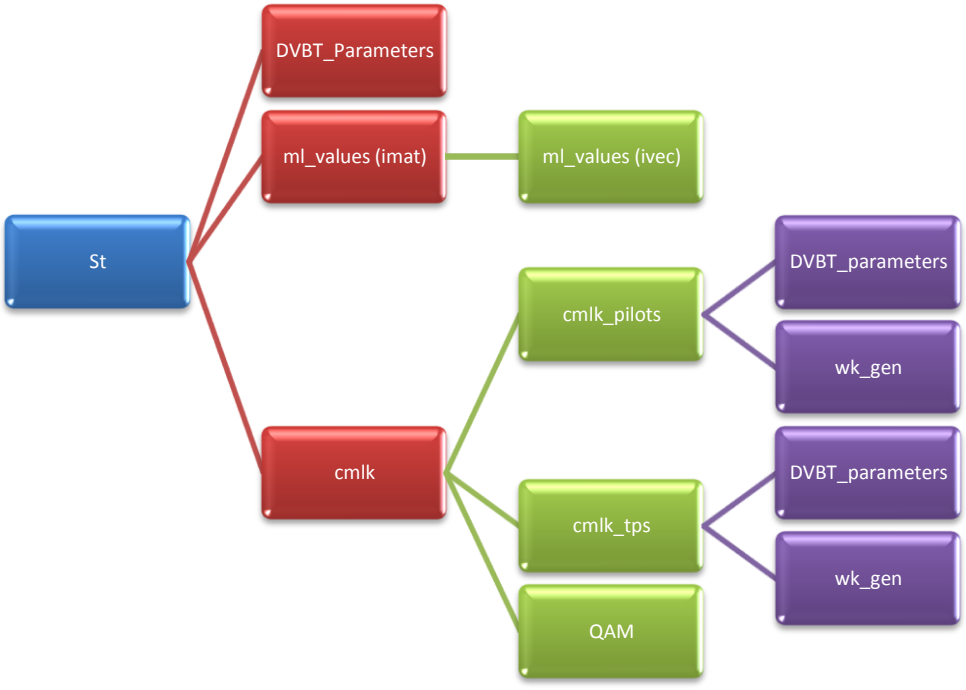

Figure 5.1.St scheme.

To create the DVB-T signal, an empty two dimension complex matrix of Kx68 positions is generated, which corresponds to a DVB-T frame ($c_{m,l,k}$). It is important to note that the elementary unit of a DVB-T signal is not a frame, but a super-frame that consists in 4 frames of 68 symbols; this procedure is repeated 4 times (it has anyways been taken into account to calculate the correct position of the pilots at the time of creating the four frames).

The positions of continual scattered and TPS pilots are calculated, and all the pilots are added after being modulated according to the reference sequence (created from the PRBS sequence). The information data is inserted in the positions where there are no pilots. The information data used is random and is generated as complex values according to the QAM modulation and the alpha parameter. The data used is random because the important information to detect the signal is the pilots. The data varies from one symbol to another and can be dismissed, but pilots are always the same.

Once the $c_{m,l,k}$ matrix is completed, values of the matrix are introduced into the expression 5.1 to create the emitted signal for each specific time.

$$s(t) = Re\left\{e^{j\,2\pi f_c t} \sum_{m=0}^{\infty} \sum_{l=0}^{67} \sum_{k=k_{min}}^{k_{max}} c_{m,l,k} \cdot \varphi_{m,l,k}(t)\right\} \qquad [5.1]$$

It has to be taken in account that in this expression, t (time) is defined as a continual parameter, but it cannot be programmed in C++ easily for a continuous range of times, so the solution used is to sample the signal. This time will be a vector of samples for a range of the times chosen and introduced in the signal as an input parameter.

Continuing in the expression, the phi signal is generated only for the first symbol and then is shifted along the rest of the symbols. This is due to the values of this signal being always the same for every symbol. This signal has also been generated for the same sampling time specified previously. The function that finally generates the emitted signal is St and it returns the real values. To be able to simulate a long signal, St must be run for a given range of times.

The received signal is represented as samples as well. As it has been explained before, it is not a continual signal due to the fact that it has been sampled in the transmitter. The sample period used is T, the elementary period, because it is a submultiple of the symbol duration and an integer number of samples can be obtained in each symbol using it.

Then two vectors are generated, one with m samples of the emitted signal for the different range of times and another with m samples of random complex values (that represent the noise). The received signal is created as a multi-path signal of 20 paths. The different paths of the channel are explained in chapter 3. Then 20 more vectors are generated (one for every path) in the same way as the one for the emitted signal, but with different delays, attenuation and phase rotation. After this, all the signal vectors obtained and the noise vector are added sample by sample. The resulting vector is the received signal which contains m samples too.

The function that generates this receceived signal is SG_DVBT and it is showed in figure 5.2.

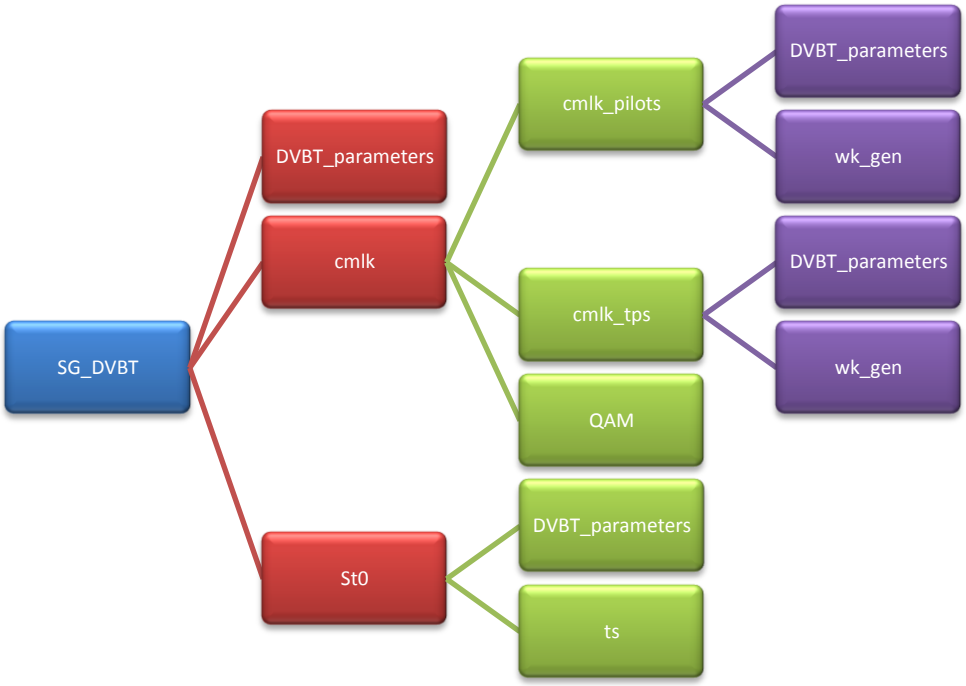

Figure 5.2.Generation of the DVB-T signal scheme.

The simulation is run for all the detectors. To compare the different sensing algorithms, a threshold value is used; as it was explained in chapter "4.3 Threshold", the procedure to obtain this value consists on introducing only noise in the detector. After running it several times for different values of random noise, a threshold value is obtained. Afterward, the statistic for the received signal is calculated after being detected. A maximum value is reached and this value is compared to the threshold to see whether the received signal contains a DVB-T signal or not. If the signal exists (H1) the maximum value will be bigger than the threshold and if there is only noise in the receptor (H0) it will be smaller.

Finally, some graphics plot the probability of detection for different values of SNR. Every new value of this SNR is obtained by changing the power of the noise which is added to the received signal. The threshold and the statistics of the received signal are recalculated for every new value of the increasing noise.

This noise is modelated as AWGN (additive white Gaussian noise). To increase the power of the noise what is really changed is its variance. The variance of the noise can be changed by varying a factor "F" that multiplies it. Therefore, increasing the noise power is nothing more than adjusting the multiplier factor "F".

For each value of this factor "F", the detector gets a maximum value and a new value of threshold is obtained. As it was explained before, these two new values are compared to check if the received signal is a DVB-T signal or not. If a signal exists, the detector returns a "1" and if the signal is absent, it returns a "0". This process is run several times (modelated in the simulation with the parameter "number of hits"). Lastly a probability of detection is calculated as the percentage of "1s" obtained in the simulation. This process is repeated for every SNR value.

Chapter 6

Simulation

The simulation performs the following steps:

- Generation of the DVB-T signal.
- Sending the DVB-T signal through the channel.
- Generation of the received DVB-T signal.
- Detection of the DVB-T signal with a specific detection algorithm.
- Repetition of the detection for a different SNR values, creating the threshold and making the comparison between signal and threshold. These simulations are done for different detectors, channels and PFAs (probability of false alarm).
- Creation of some graphics in terms of probability of detection for all the detection algorithms.

All the simulations have been done with the following parameters to generate the DVB-T signal:

- **BW** (*bandwidth*): 8 MHz.
- **Mode** (*mode of operation*): 2K.
- **GI** (*guard interval*): 2, corresponding to $^1/_4$. The duration of the symbol is: $224\ \mu s$
- **Level** (*QAM constellation*): 4 for 16 QAM (4 bits).
- **Alpha**: 2 for hierarchical.
- **Cr1, Cr2** (*code rate*): $^3/_4$ for Cr1 and $^5/_6$ for Cr2.
- **RF** (*frequency*): 0. so, the emission is in base band.
- **T:** 0.01792 (generation of 80 symbols).

Other important parameters to take into account are:

- **PFA:** 0.1, 0.01 and 0.001. Normally 0.1 and 0.01. The 0.001 is only for one simulation to see the impact.
- **Number of hits:** 100 (number of simulation for every SNR value)**.**
- **Max number of simulations:** 2.000**.**
- **Snr_dB:** From -40 to -10 dB with separation of 1dB between simulations.
- **Channel:** AWGN, Rayleigh and Rician (20 paths).

Once all the parameters are choosen, the simulation is run for each detector. For each simulation, all the important information is saved in a file (.it). The variables used in this file are B (Bandwith), CP (value of the cyclic prefix), PD (vector with the probability of detection), PFA (probability of false alarm), mode and SNR (vector with all the values of SNR simulated). Based on this data, graphs are generated.

6.1. Energy Detector

The energy detector is the first sensing algorithm which is going to be analysed. As it was explained in chapter 4, it is one of the simplest detectors that can be constructed.

First simulation

In this first simulation, the energy detector has been simulated with a PFA of 0.1 and in an AWGN channel (no multi-path). The idea is to observe the behavior of the probability of detection depending on the SNR (measure used in engineering to quantify how much a signal has been corrupted by noise ($SNR = \frac{P_{signal}}{P_{noise}}$)). Furthermore, the energy detector simulated is compared with the values obtained theorically.

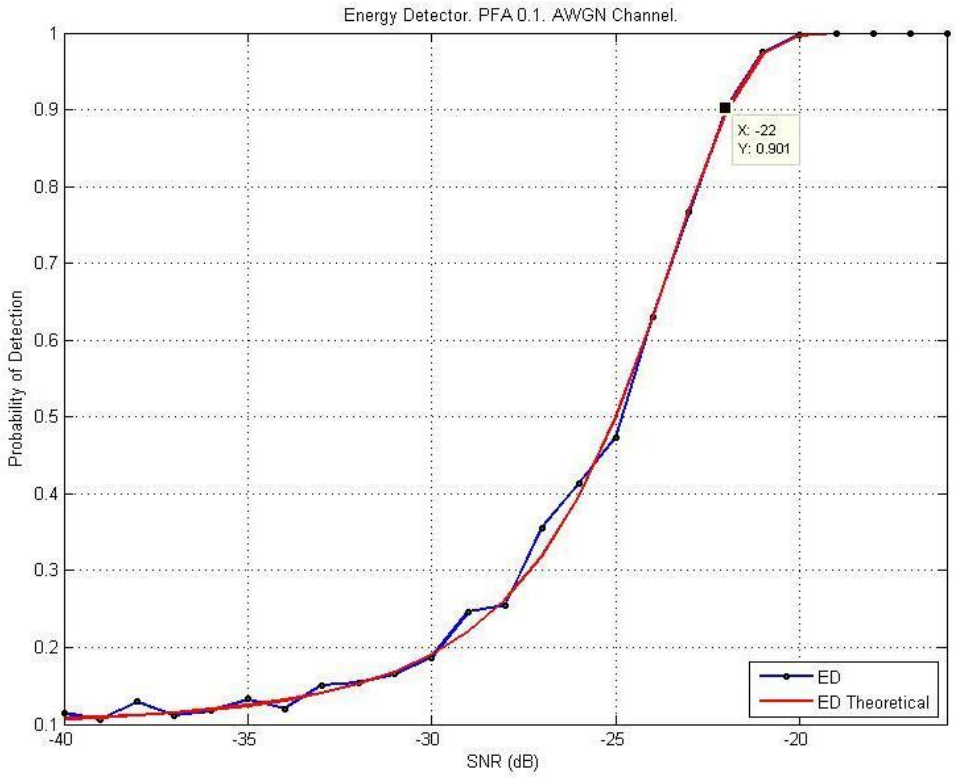

Figure 6.1 Energy Detector, PFA = 0.1, AWGN channel.

As expected the probability of detection decreases exponentially for decreasing SNR values. For a probability of detection of 90% the SNR needed is around -22 dB. This SNR value can be considered a good result. Also, the behaviour of the energy detector simulated fits very well with the theorically calculated values. Finally it can be observed that the probability of detection changes very softly in the transition range (from 0.9 to 0.2) in approximately 8 dB (-22 dB to 90% and -30 dB to 0.2%).

Second simulation

In this second simulation, the energy detector has been simulated with a PFA of 0.01 and in an AWGN channel (no multi-path). The idea is to observe which changes occur when the values of PFA are changed.

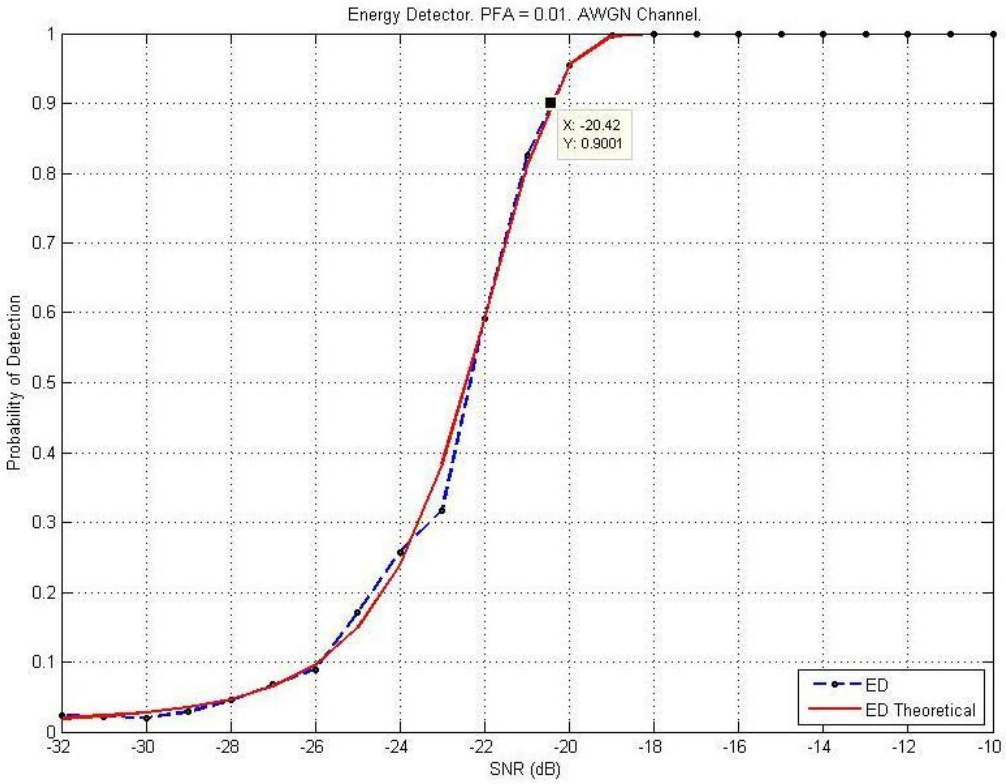

Figure 6.2 Energy Detector, PFA = 0.01, AWGN channel.

As in the previous graph, the behaviour of the energy detector simulated fits very well with the theorically calculated values. For a probability of detection of 90% the SNR needed is around -20.4 dB. This SNR value still can be considered a good result. It can be observed that the probability of detection changes in the transition range in approximately 3.9 dB (-20.4 dB to 90% and -24.3 dB to 0.2%).

Third simulation

In this third simulation, a comparative between graphs 6.1 and 6.2 is done. The energy detector has been simulated with a PFA of 0.1 vs 0.01 in an AWGN channel (no multi-path). The idea is to clearly visualize the difference after changing the value of the PFA.

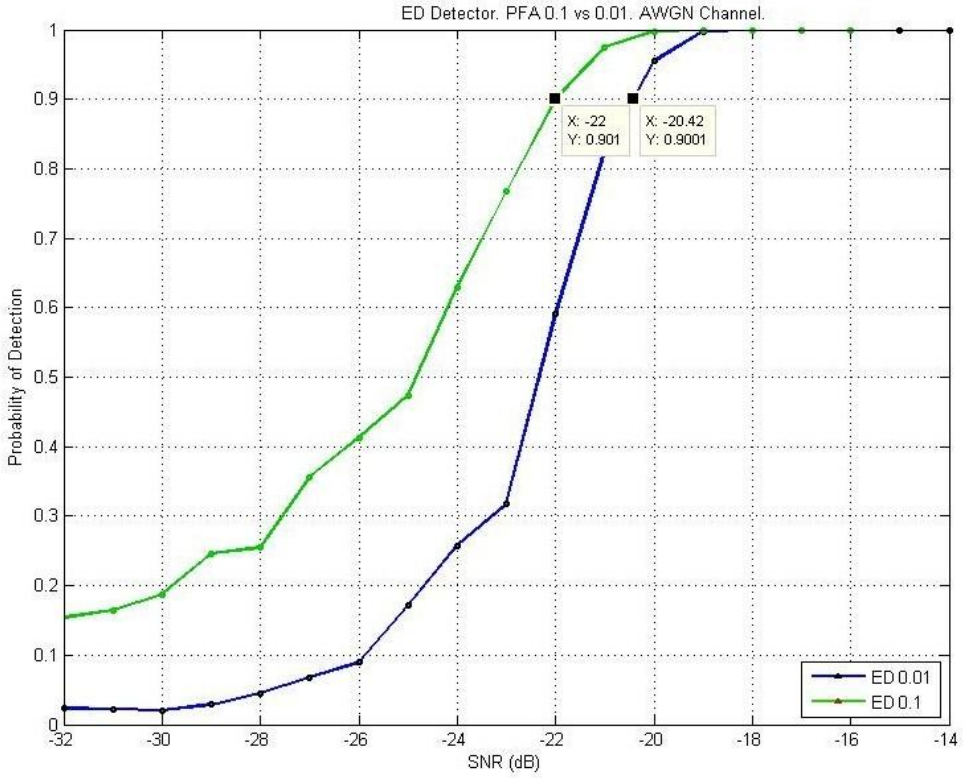

Figure 6.3 Energy Detector, PFA = 0.1 vs 0.01, AWGN channel.

It could be appreciated that the behavior of the performance of the energy detector when it is working with a PFA value of 0.1 is much better than when is working with a PFA value of 0.01. In the first case, a 90% of probability detection is reached for a SNR of -22dB, while in the second case is reached for a SNR of -20.4. At first sight it doesn't seem to be very significant (just 1.6 dB), but if it is looked carefully, you realize that for a SNR of -22 dB, the probability of detection in the second detector falls to 60% (90% in the first one). For SNR of -20.4 the first detector has a probability of detection near 99.5% (90% in the second one), so the difference is substancial.

Another clear difference is that the probability of detection changes in the transition range by 8 dB (-22 dB to 90% and -30 dB to 0.2%) in the first case and by 3.9 dB in the second one (-20.4 dB to 90% and -24.3 dB to 0.2%). Finally, for a SNR of -24.3, the probability of detection of the second detector is 20%, while in the first detector is 50%.

Fourth simulation

In this fourth simulation, the energy detector has been simulated with a PFA of 0.1, 0.01 and 0.001 and in all three different channels (AWGN, and multipath channels Rayleigh and Rician). The idea is to observe all possible changes when PFA vary and the behavior of the detectors in a multipath channel.

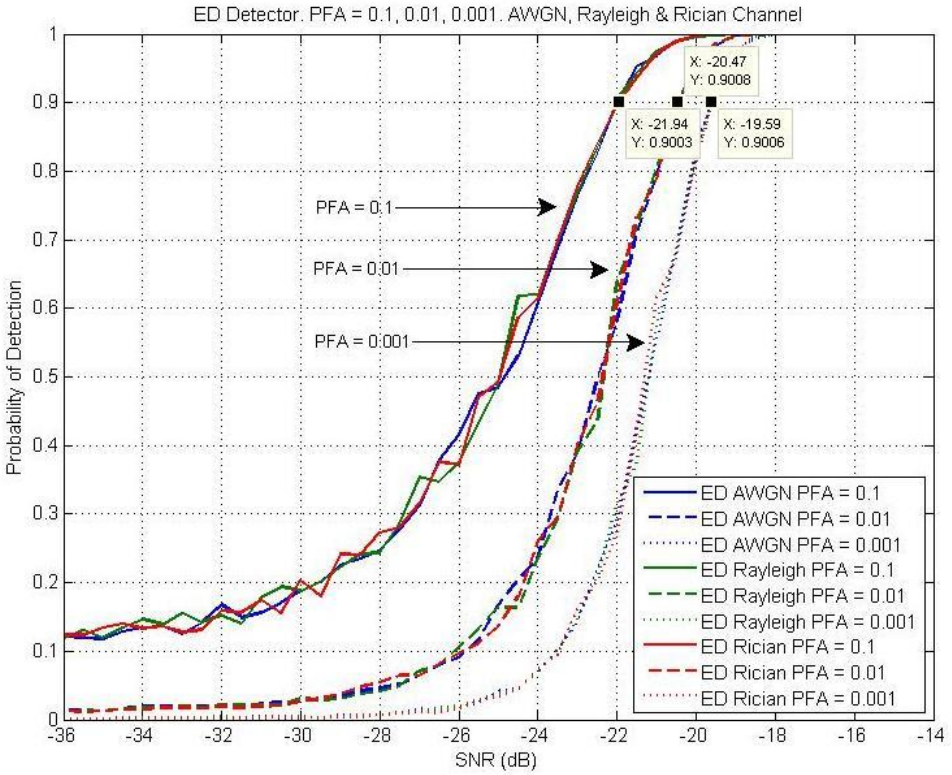

Figure 6.4 Energy Detector. PFA = 0.1, 0.01, 0.001. AWGN, Rayleigh & Rician channel.

In this graph, three important conclusions can be obtained.

1. The smaller the value of PFA, the worst the detection probability is.
2. The smaller the value of PFA, the sharper the slope in the transition range.
3. The behavior of the detector in different channels is more or less the same, so that great variations on the behavior can not be appreciated.

For a detector with PFA of 0.1, the 90% of probability detection is reached for a SNR of -22dB; for a PFA of 0.01 is reached for a SNR of -20.4 dB and for a PFA of 0.001 is reached for a SNR of -19.6 dB. But the important is that for a value of SNR of -22 dB, the detector with PFA of 0.1 has a 90% of probability of detection, while for a PFA of 0.01 it decreases until 60% and for a PFA of 0.001 decreases lot more to 30%.

As it is commented in the third conclusion, no major variations are observed in the graphs after crossing an AWGN channel, a Rayleigh channel or a Rician channel. All three take similar values of probability of detection, so the behavior is very similar in one channel or in other.

Despite that the simulations have been developed for all channels and for all possible combinations of PFA, from now, only the AWGN channel simulation are going to be shown. The reason is because the behavior is very siminar in all channels and showing them doesn't add information.

Fifth simulation

In this fifth simulation, the graphic is the same as shown in the previous simulation, but probability of detection is shown in dB. The idea is to find some proportionality between curves and see if something new appears.

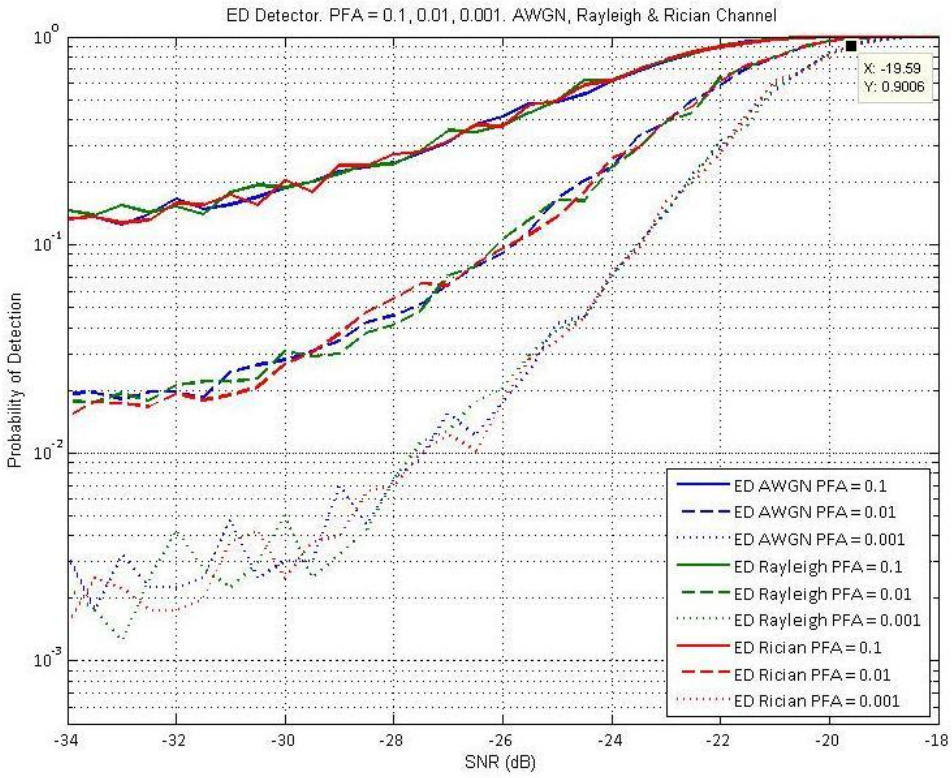

Figure 6.5 Energy Detector. PFA = 0.1, 0.01, 0.001. AWGN, Rayleigh & Rician channel. PD (dB).

As I suspected, another conclusion can be obtained:

1. As SNR tends to minus infinity, probability of detection tends to reach a saturation value which coincides with the probability of false alarm.

6.2. Autocorrelation Detector

Once the graphs of the energy detector have been analysed, the next step is to analyse the autocorrelation detector and see if the behavior is similar or not to the previous one. The autocorrelation coefficient detector is a simple and computationally efficient spectrum sensing scheme for OFDM based on its autocorrelation coefficient.

Sixth simulation

In this sixth simulation, the autocorrelation detector has been simulated with a PFA of 0.1 and in an AWGN channel (no multi-path). The idea is to observe if the behavior is similar to the energy detector in term of values or they are very diffent.

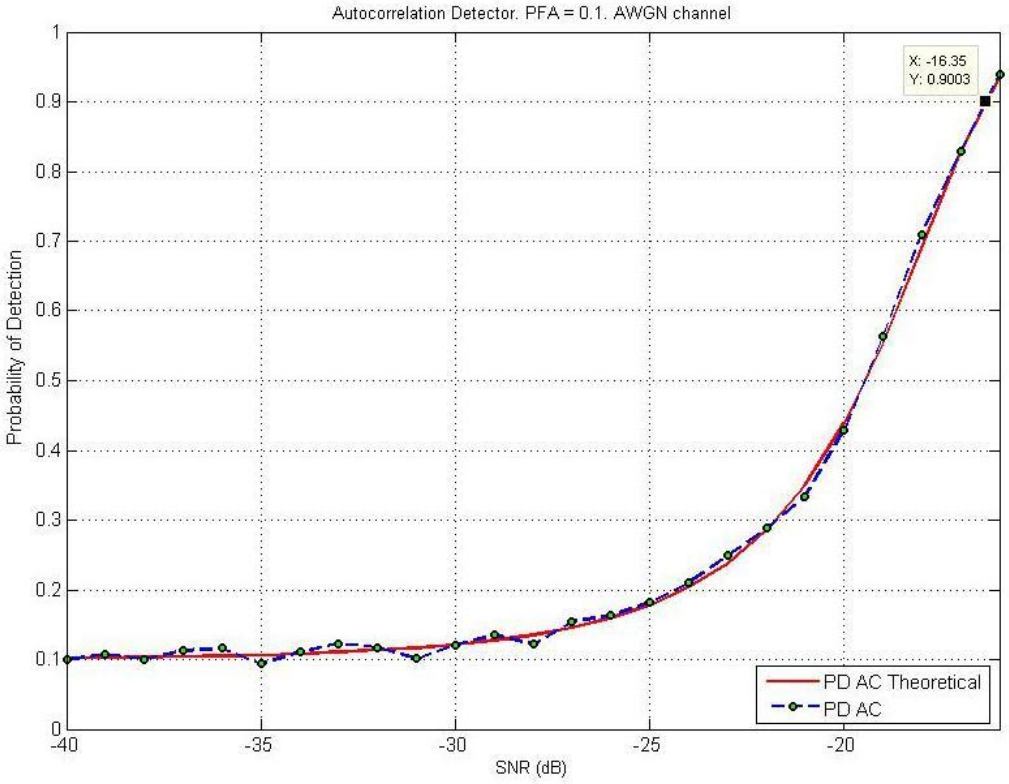

Figure 6.6 Autoccorrelation Detector, PFA = 0.1, AWGN channel.

As it was expected, the shape is very similar to the energy detector although the results obtained are considerably worse. For a probability of detection of 90% the SNR needed is around -16.4 dB (the previous one was -22 dB). So, for a SNR of -22 dB, the probability of detection in this detector falls to 28% (90% in the first one).

It can be observed that the probability of detection changes very softly again in the transition range in approximately 7.6 dB (-16.4 dB to 90% and -24.0 dB to 0.2%). Finally the data obtained in the simulation is very close to those calculated theoretically.

Seventh simulation

In this seventh simulation, the autocorrelation detector has been simulated with a PFA of 0.01 and in an AWGN channel (no multi-path). The idea is to observe if the behavior with the autocorrelation detector is similar as the energy detection with parameters 0.01.

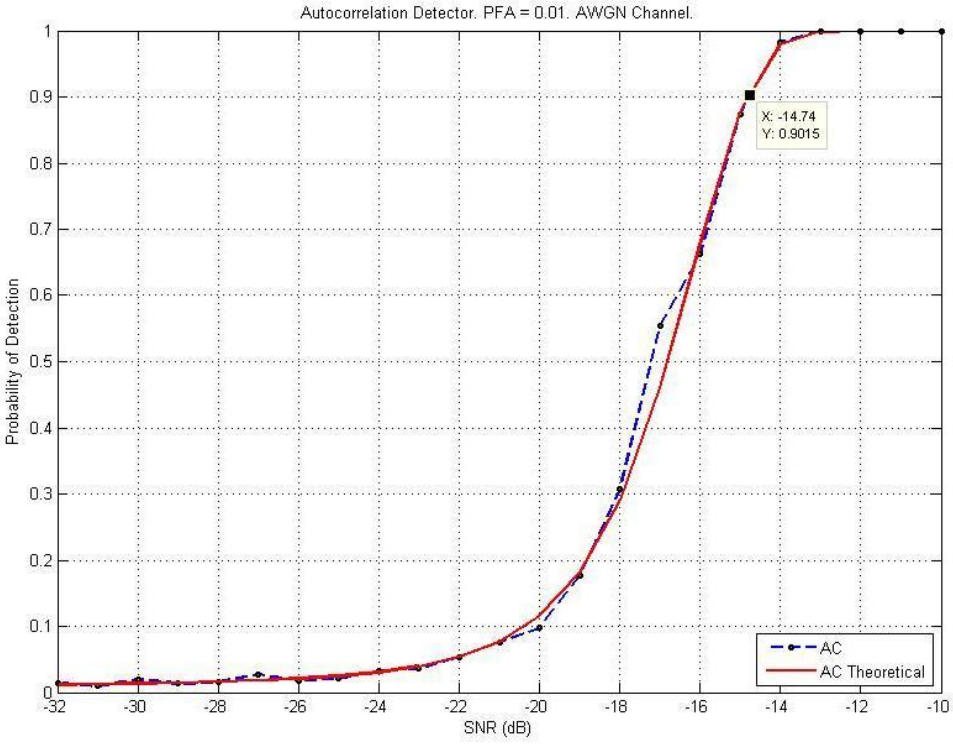

Figure 6.7 Autoccorrelation Detector, PFA = 0.01, AWGN channel.

For a probability of detection of 90% the SNR needed is around -14.7 dB. It can be observed that the probability of detection changes in the transition area in approximately 4.2 dB (-14.7 dB to 90% and -18.9 dB to 0.2%). Also, the data obtained in the simulation is very close to that which was calculated theoretically.

Eighth simulation

In this eighth simulation, a comparative between graphs 6.6 and 6.7 is done. The autocorrelation detector has been simulated with a PFA of 0.1 vs 0.01 and in an AWGN channel (no multi-path). The

idea is to observe if the behavior with the autocorrelation detector is similar to what happened between the energy detection with parameters 0.1 and 0.01.

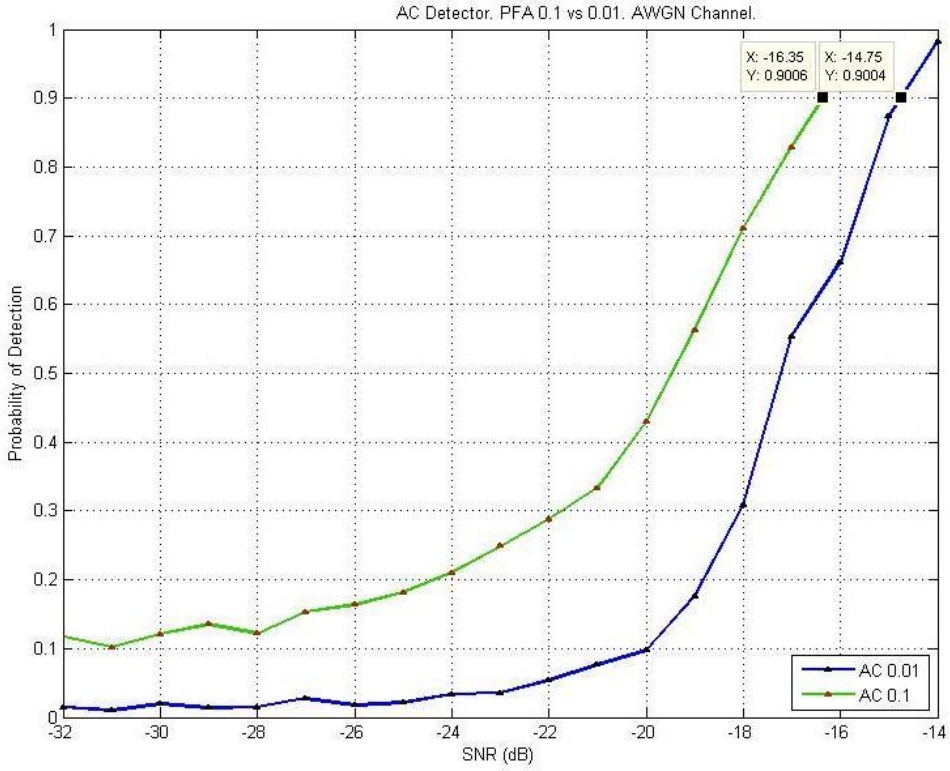

Figure 6.8 Autoccorrelation Detector, PFA = 0.1 vs 0.01, AWGN channel.

It could be appreciated that the behavior of the performance of the energy detector when it is working with a PFA value of 0.1 is much better than when is working with a PFA value of 0.01 as in the energy detector. In the first case, a 90% of probability detection is reached for a SNR of -16.35 dB, while in the second case is reached for a SNR of -14.75 dB (about 1.6 dB less). For a SNR of -16.35 dB, the probability of detection in the second detector falls to 65% (90% in the first one).

Another difference is that the probability of detection changes in the transition range in approximately 7.6 dB (-16.4 dB to 90% and -24 dB to 0.2%) for a PFA of 0.1 and by 4.2 dB for a PFA of 0.01 (-14.7 dB to 90% and -18,9 dB to 0.2%). Finally, for a SNR of -18.9, the probability of detection of the AC for PFA of 0.01 is 20%, while for PFA of 0.1 is 60%.

6.3. Cyclic Prefix Based Sliding Correlation

By this point, energy detector and autocorrelation detector have been analysed with curves very similar but with so different values. The initiative is to obtain the results for the CP detector and

determine if is still related with the others or exist some clear differences. Two different cyclic prefix detectors are studied, the real and the absolute one.

Ninth simulation

In this simulation, a comparison of four different graphs is made. The Cyclic Prefix Based Sliding Correlation detector has been simulated for real and for absolute values with a PFA of 0.1 vs 0.01 and in an AWGN channel (no multi-path). The idea is to observe if the behavior with the cyclic prefix detector is similar to what happened between the previous two detectors.

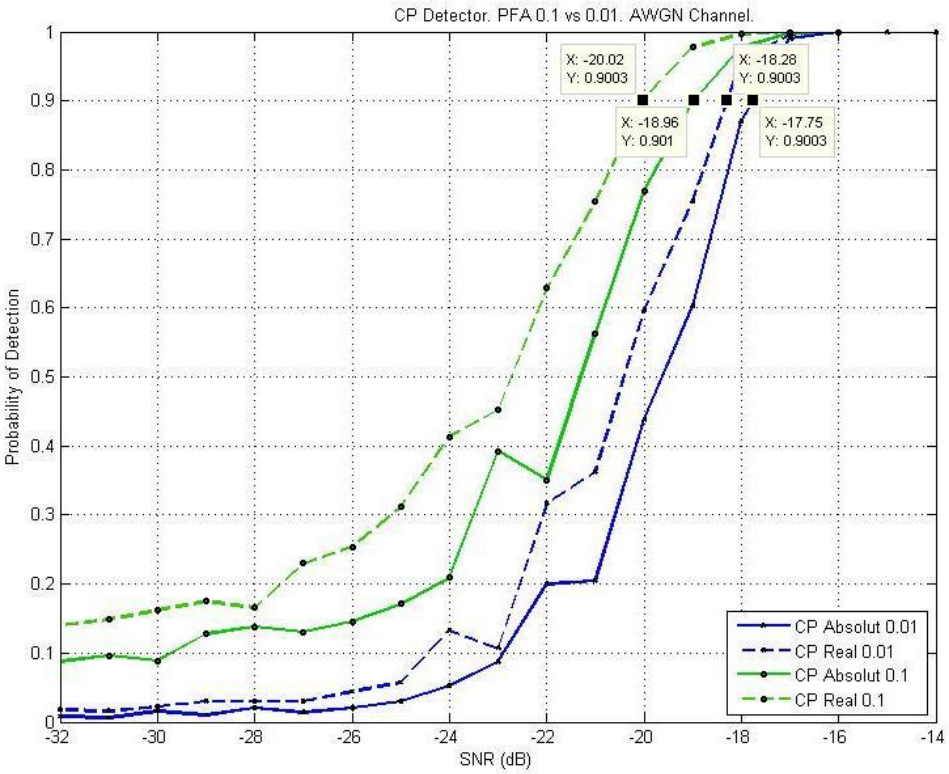

Figure 6.9 CP Detector, PFA = 0.1 vs 0.01, AWGN channel.

As it could be seen, the behavior of the real CP detector is slightly better than the absolute CP detector. Furthermore, as in the others detectors, exist a clear difference between the simulations made with PFA = 0.1 and PFA = 0.01.

For the real CP detector, a 90% of probability detection is reached for a SNR of -20dB, while for the absolute CP detector is reached for a SNR of -19 dB. It is just only 1 dB of difference between them.

In terms of probability of detection, it can be observed that for a SNR of -20 dB, the probability of detection of absolute CP detector is near 78 % (90% in the real CP), the difference is not as important, although it is obviously better.

Tenth simulation

In this tenth simulation, the CP detector has been simulated with a PFA of 0.01 and in all three different channels (AWGN, and multipath channels Rayleigh and Rician). The idea is to observe if there is any difference between the detection on each channel.

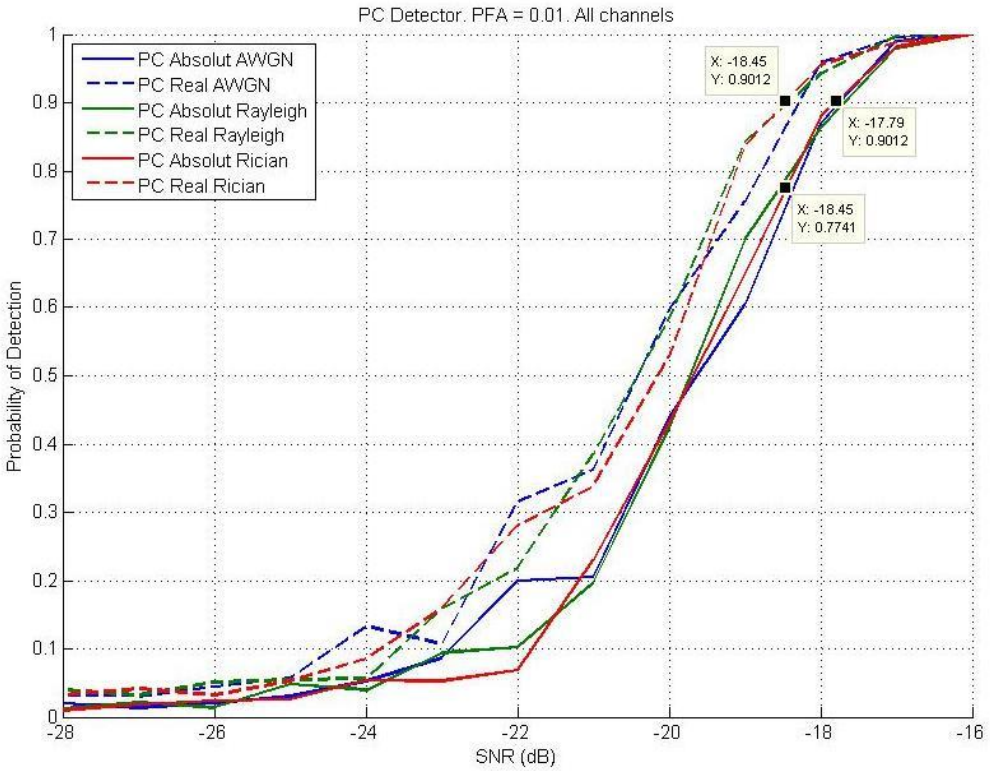

Figure 6.10 PC Detector, PFA = 0.01, All channels.

As it was commented in the forth simulation, no big variations are observed in the curves after crossing an AWGN channel, a Rayleigh channel or a Rician channel. As expected, the three curves show a similar behavior in terms of probability of detection.

6.4. Time Domain Pilot Based Sliding Correlation

Right now, time domain pilot based sliding correlation simulations is going to be analysed. As it was explained in chapter 4, the statistic obtained with this algorithm is similar as the cyclic prefix based

sliding correlation but in this case, the second term only contains the pilots. Let's see if this implies a strong improvement.

Eleventh simulation

In this simulation, a comparative of four different graphs is made once again. The time domain pilot based sliding correlation detector has been simulated for real and for absolute values with a PFA of 0.1 vs 0.01 and in an AWGN channel (no multi-path).

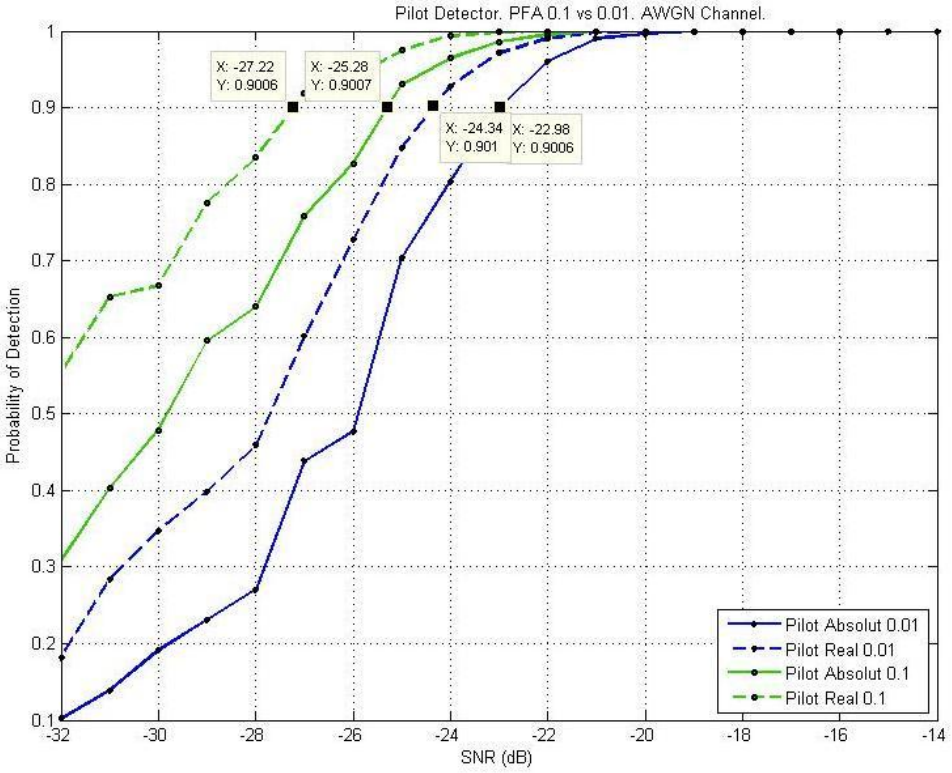

Figure 6.11 Pilot Detector, PFA = 0.1 vs 0.01, AWGN channel.

As it can be seen at a glance, the behavior of the real Pilot detector is visibly better than the absolute Pilot detector. For the real Pilot detector, a 90% of probability of detection is reached for a SNR of -27.2 dB. This SNR value can be considered a very good result and it is the best of the detectors studied until now.

For the absolute Pilot detector, a 90% of probability of detection is reached for a SNR of -25.3 dB (about 2 dB more than the real one). If you look in terms of probability of detection, it could be observed that for a SNR of -27.2 dB, the probability of detection of absolute Pilot detector is near

75% (90% in the real Pilot). Furthermore, as in the others detectors, there exist a clear difference between the simulations made with PFA = 0.1 and PFA = 0.01.

6.5. Time domain Pilot in Cyclic Prefixes Based Sliding Correlation

In the case of Cyclic Prefix Pilot Detector all the simulations have been done, but after analyzing the acquired data I have arrived to the conclusion that the results obtained were not satisfactory and they can not be considered reliable, hence they were discarded due to the fact that they showed abnormal behavior.

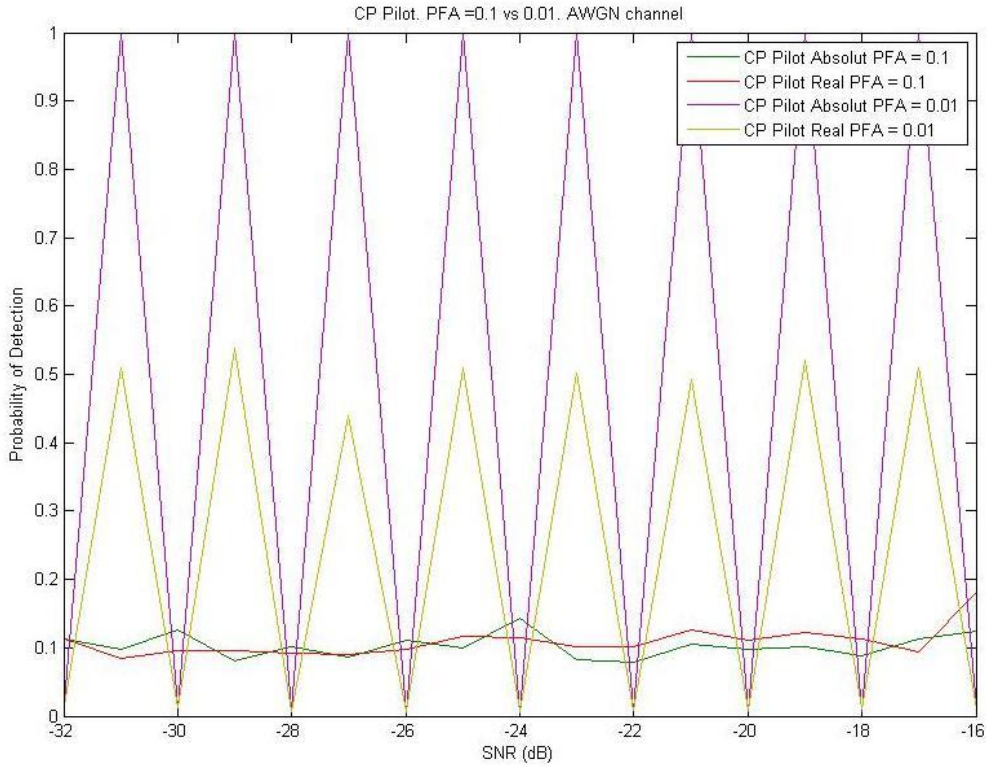

Figure 6.12 CP_Pilot Detector, PFA = 0.1 vs 0.01, AWGN channel.

The C++ code has revised exhaustively to find the possible error but after some time and different tests a solution has not been found. Anyway, according to a previous study, results should be very similar to the ones achieved with the CP detector.

6.6. All detectors

Finally all the detectors have been studied and it is time to compare all of them together to see the differences between each one more clearly.

Twelfth simulation

In this twelfth and last simulation, all the detectors have been simulated with a PFA of 0.1 and in an AWGN channel (no multi-path). The idea is to be able to compare them easily.

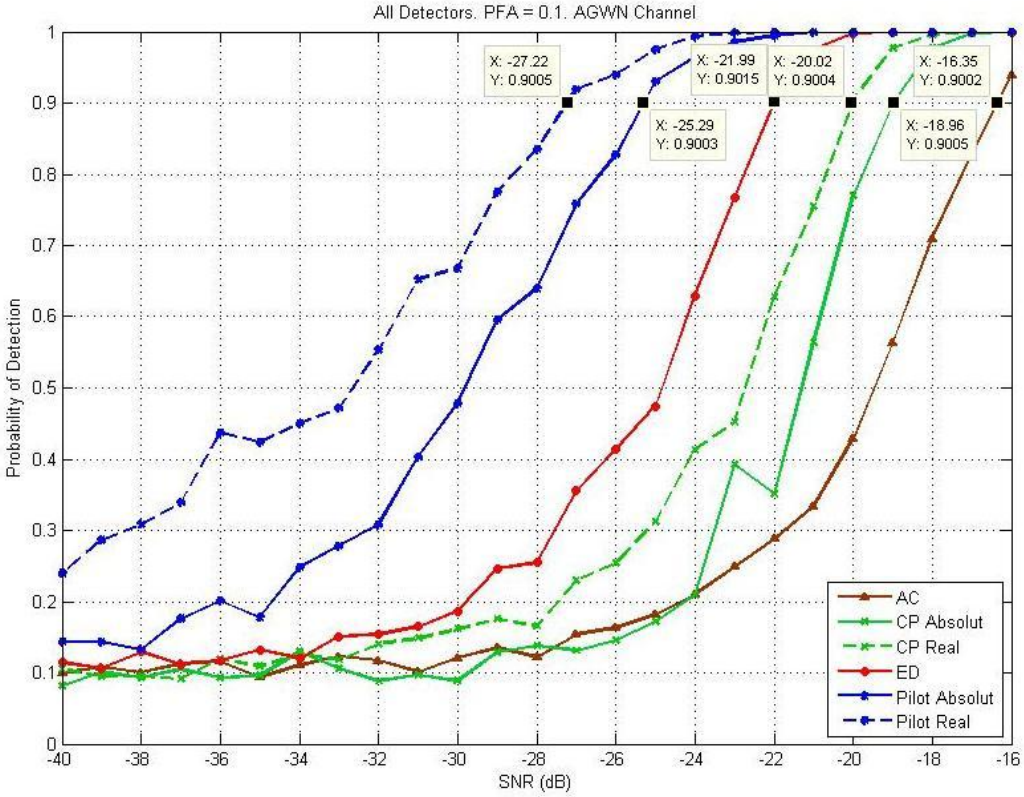

Figure 6.13 All Detectors, PFA = 0.1, AWGN channel.

A table with all the values of probability of detection is shown below:

	ED (dB)		AC (dB)		CP abs (dB)		CP real (dB)		Pilot abs (dB)		Pilot real (dB)	
PFA	0.1	0.01	0.1	0.01	0.1	0.01	0.1	0.01	0.1	0.01	0.1	0.01
90% PD	-22	-20.4	**-16.4**	-14.7	-19	-17.8	-20	-18.3	-25.3	-23	**-27.2**	-24.3
20% PD	-30	-24.3	-24	-18.9	-24	-21	-27.6	-22.7	-34.8	-30	-42	-32
Diference	8	3.9	7.6	4.4	5	**3.2**	7.6	4.4	9.5	7	**14.8**	7.7

Table 6.1.Probability of detection of all sensing algorithms

As it could be observed in Table 6.1 and figure 6.12, the best performing detector is the real Pilot detector. For it, a 90% of probability of detection is reached for a SNR of -27.2 dB (about 2 dB more than the absolute one, 5dB more than the ED and 7 dB more than CP).

It should be noticed that for a SNR of -27.2 dB, the probability of detection for absolute Pilot detector is near 75% (90% in the real Pilot), for energy detector is near 35% while for the rest is less than 25%.

Finally, figure 6.13 shows the simulation of all the detectors with a PFA of 0.01 and in an AWGN channel (no multi-path).

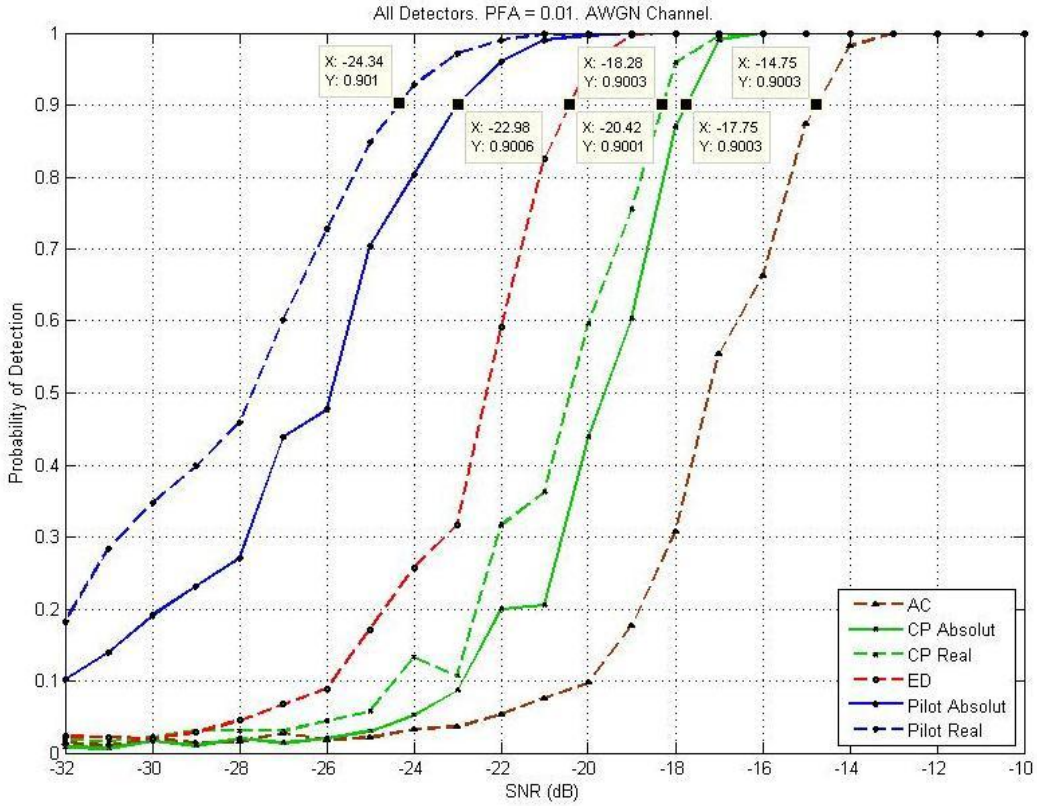

Figure 6.14 All Detectors, PFA = 0.01, AWGN channel.

Chapter 7

Conclusions

For the realization of this study, five pilot sensing algorithms have been tested; the algorithms are based on two pilot basic propierties: Cyclic prefix (CP) and pilot symbols, which permit the detection of the DVB-T signal in an efficient way. A C++ application has been developed to simulate the DVB-T transmission. After obtaining the simulation data and analysis, a number of conclusions can be drawn about the algorithms:

- **Time Domain Pilot Based Sliding Correlation:** This is the best algorithm tested of all five. Its behavior is excellent for obtaining high values of SNR for a probability of detection. It is a bit more complex than the others because it needs to obtain TPS information before hand, but the time needed to make the simulations was shorter than I expected. Undoubtedly the best option in most cases.

- **Energy Detection (Power detection):** It has been the next best performing algorithm, but probably is the simplest detector that can be constructed in practice. If the computational requirements are low, it may be the best option.

- **Cyclic Prefix Based Sliding Correlation:** It has been the third best performing algorithm with better results in term of probability of detection, but simulations take too long time.

- **Autocorrelation Coefficient Detector:** Altough it was supposed to be an effective and computationally efficient, the time needed to make the simulations is not as short as was expected. The results obtained in term of probability of detection are really bad in comparison with the others.

- **Time domain Pilot in Cyclic Prefixes Based Sliding Correlation:** The results obtained were not as satisfactory as expected and they were discarded due to show abnormal behavior. Anyway, according to a previous study, results should be very similar to the ones achieved with the CP detector. Time domain cyclic prefixes based sliding correlation is a bit more complex but takes less time of computational requierements.

Furthermore, some additional consideration about the study of the probability of false alarm should be regarded:

1. The smaller the value of PFA, the worse the detection probability is.
2. The smaller the value of PFA, the bigger the slope in the transition range.
3. As SNR tends to minus infinity, probability of detection tends to reach a saturation value which coincides with the probability of false alarm.

And finally, the behavior of the detectors in different channels is more or less the same, so that great variations on the behavior can not be expected.

Appendix A

Detailed explanation of the C++ functions

In this appendix, a wide explanation of the functions programmed is done for a better understanding of the code and the complete application.

Fading_Parameters

Description:

This function models the behavior of the channel. It calculates the channels transfer function from three basic parameters defined for a multipath channel model such as time delay (phi), power delay (taul) and phase rotation (rho) as defined in Chapter 3 "Channel model". Three different kind of channel are modelated: Rayleigh, Rician that are multipath ones and AGWN.

The expressions used to define these two multipath channels are:

- Rayleigh

$$y(t) = \frac{\sum_{i=1}^{N} \rho_i e^{-j\theta_i} x(t-\tau_i)}{\sqrt{\sum_{i=1}^{N} \rho_i{}^2}} \qquad [A.1]$$

- Ricean:

$$y(t) = \frac{\rho_0 x(t) + \sum_{i=1}^{N} \rho_i e^{-j\theta_i} x(t-\tau_i)}{\sqrt{\sum_{i=1}^{N} \rho_i{}^2}} \qquad [A.2]$$

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
String	Type	Define the kind of channel to simulate: "Rayleigh", "Rician" or "AWGN"
Cvec&	Hl	Modeled channel
vec&	Taul	Values of the time delay parameters

Table A.1 Input Fading_parameters

DVBT_parameters

Description:

In this function all the parameters needed to generate the DVB-T emitted signal are obtained. These are: BW, mode, GI, T, Tu, Delta, TS, Kmin, Kmax, K y N. The expressions to calculate them are:

- $T = \frac{7 \cdot 10^{-6}}{8 \cdot BW}$ [A.3]
- $T_u = 1024 \cdot mode \cdot T$ [A.4]

- $Delta = T_u \cdot 2^{-GI} = 1024 \cdot mode \cdot T \cdot 2^{-GI}$ [A.5]
- $TS = T_u + Delta$ [A.6]
- $K_{max} = 852 \cdot mode$ [A.7]
- $K = K_{max} - K_{min} + 1$ [A.8]
- $N = (1024 + 2^{10-GI}) \cdot mode$ [A.9]

Inputs & Outputs:

INPUTS			
Type	**Variable**	**Description**	
Ivec	DVB	DVB(0) = BW	Bandwidth 6, 7, 8 (MHz)
		DVB(1) = Mode	Mode of operation 2 (2k mode) or 8 (8k mode)
		DVB(2) = GI	Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)
Vec &	Times	Times(0) = T	Sampling interval in seconds
		Times(1) = Tu	Useful part duration
		Times(2) = Delta	Guard interval duration
		Times(3) = TS	Total symbol duration
Ivec &	Pars	Pars(0) = Kmin	First carrier number
		Pars(1) = Kmax	Last carrier number
		Pars(2) = K	Number of carriers per symbol
		Pars(3) = N	Number of samples per symbol

Table A.2 Input DVBT_parameteres

QAM

Description:

Generation of a complex vector of coefficients with the constellation coordinates. It is generated from a binary sequence which is the information data received as input. It depends on two parameters: alpha and level.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Bvec	InVec	Binary sequence

Ivec	modp	modp(0) = Level Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6 2 for QPSK (2 bits) 4 for 16 QAM (4 bits) 6 for 64 QAM (6 bits) modp(1) = alpha Proportions of the constellation. 0 (non-hierarchical) 1, 2 or 4 (hierarchical)

OUTPUTS		
Type	**Variable**	**Description**
Cvec	Loc_vec	Returning a complex vector of nr_symbols dimension which all the data mapped. (Comment: nr_symbols = InVec.length() / level)

Table A.3 Input & Output QAM

wk_gen

Description:

Generation of the reference sequence from the PRBS sequence (wk) modulated for scattered and continual pilots. It is generated from a LFSR with generator polynomial $X^{11} + X^2 + 1$ and initial seed all to one. More information can be found in "2.5.1 Definition of reference sequence". Also the reference sequence is modulated according this expression (that corresponds to continual and scattered pilot modulation):

$$\mathrm{Re}\{c_{m,l,k}\} = \frac{4}{3} \cdot 2 \cdot (\frac{1}{2} - w_k) \qquad \mathrm{Im}\{c_{m,l,k}\} = 0 \qquad [A.10]$$

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Int	Nr_carriers	Number of carriers for wk

OUTPUTS		
Type	**Variable**	**Description**
Cvec	Loc_vec	Returning a complex vector of nr_carriers dimension with the modulation of the

		PRBS sequence.

Table A.4 Input & Output wk_gen

ml_values (ivec & imat)

Description:

Generate a vector with the values of l (symbol) and l (frame) for a given "t" (in ivec), or for a given range of "t" (in imat).

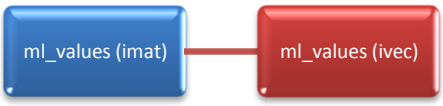

Figure A.1.ml_values scheme.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
double	Ts	Binary sequence
double (ivec) Vec (imat)	t	(ivec) Instant time to calculate the values of l and m (imat) Vector of times to calculate the values of l and m

OUTPUTS		
Type	**Variable**	**Description**
Ivec imat	Loc_vec Loc_imat	Returning the values of l (symbol) and m (frame), for an instant time (ivec) or for a vector of times (imat)

Table A.5 Input & Output ml_values

cmlk_pilots

Description:

Generate a matrix with the continual and scattered pilots. For this, first a matrix of 68·K is created, the reference sequence is calculated and first continual and then scattered pilots are added. The position of the continual pilots is given and the position of scattered pilots is calculated as it shows the next expression:

$$k = K_{min} + 3 \cdot (l \bmod 4) + 12p \qquad p \in int, p \geq 0 \quad k \in [K_{min} : K_{max}] \qquad [A.11]$$

Finally, the matrix with both pilots is returned.

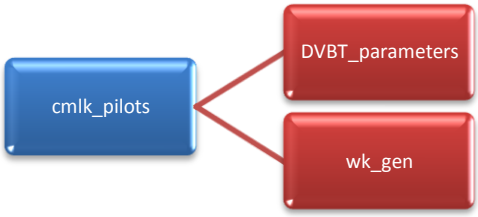

Figure A.2.cmlk_pilots scheme.

Inputs & Outputs:

INPUTS			
Type	**Variable**	**Description**	
Ivec	DVB	DVB(0) = BW	Bandwidth 6, 7, 8 (MHz)
		DVB(1) = Mode	Mode of operation 2 (2k mode) or 8 (8k mode)
		DVB(2) = GI	Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)

OUTPUTS		
Type	**Variable**	**Description**
cmat	Loc_cmat	Return a matrix of 68·k with the continual and scattered pilots.

Table A.6 Input & Output cmlk_pilots

cmlk_tps

Description:

Generate a matrix with the tps pilots. For this, first a null matrix of 68·K is created, the reference sequence and position is given. A vector of 68 components is created and all the TPS information is inserted in it. Finally, according to "2.5.4.2 TPS modulation" the tps pilots are modulated and inserted in the positions in the matrix. The expression of TPS modulation is showed below:

$$Re\{c_{m,l,k}\} = 2 \cdot ({}^{1}/_{2} - w_k); \qquad Im\{c_{m,l-1,k}\} = 0; \qquad \text{[A.12]}$$

Finally, the matrix with TPS pilots is returned.

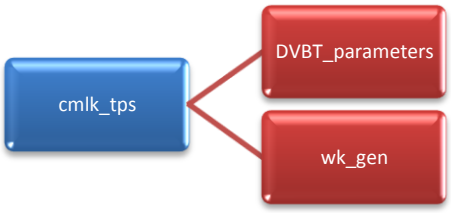

Figure A.3.cmlk_tps scheme.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Int	m	Frame in the super frame. 1,2,3 or 4
Ivec	DVB	DVB(0) = BW Bandwidth 6, 7, 8 (MHz) DVB(1) = Mode Mode of operation 2 (2k mode) or 8 (8k mode) DVB(2) = GI Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)
Ivec	modp	modp(0) = Level Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6 2 for QPSK (2 bits) 4 for 16 QAM (4 bits) 6 for 64 QAM (6 bits) modp(1) = alpha Proportions of the constellation. 0 (non-hierarchical)
vec	codep	Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then cr2 = 0. codep(0) = cr1 codep(1) = cr2

OUTPUTS		
Type	**Variable**	**Description**
cmat	Loc_cmat	Return a matrix of 68·k with the tps pilots.

Table A.7 Input & Output cmlk_tps

cmlk

Description:

Generate a matrix with all the pilots (continual, scattered and TPS pilots) and modulated random data. First continual and scattered pilots are inserted in a null 68·K matrix. Then TPS pilots are added to the previous matrix. A vector of Bernoulli random data is generated and modulated according to level and alpha. Finally, the modulated random data generated is added to the pilots' matrix, in the position where there are no pilots. . Finally, the matrix (cmlk) is returned. Also a second version exists, where data is not random and it is obtained as an input (bvec data).

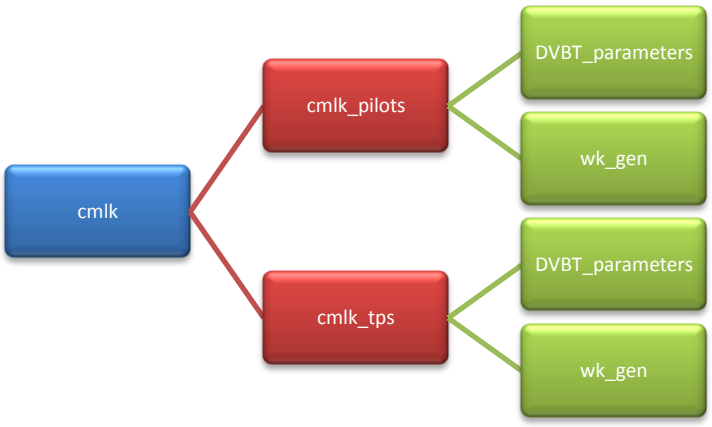

Figure A.4. cmlk scheme.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Int	m	Frame in the super frame. 1,2,3 or 4
Ivec	DVB	DVB(0) = BW — Bandwidth 6, 7, 8 (MHz) DVB(1) = Mode — Mode of operation 2 (2k mode) or 8 (8k mode) DVB(2) = GI — Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)
Ivec	modp	modp(0) = Level — Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6 2 for QPSK (2 bits) 4 for 16 QAM (4 bits) 6 for 64 QAM (6 bits) modp(1) = alpha — Proportions of the constellation. 0 (non-hierarchical)
Vec	codep	Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then cr2 = 0. codep(0) = cr1 codep(1) = cr2

OUTPUTS		
Type	**Variable**	**Description**
cmat	Loc_cmat	Return a matrix of 68·k with the all the pilots (continual, scattered and TPS pilots) and modulated random data.

Table A.8 Input & Output cmlk

st

Description:

Generate a vector with the ST signal for a given number of samples of "t". The expression to obtain this signal is showed below.

$$s(t) = Re\left\{e^{j\,2\pi f_c t} \sum_{m=0}^{\infty} \sum_{l=0}^{67} \sum_{k=k_{min}}^{k_{max}} c_{m,l,k} \cdot \varphi_{m,l,k}(t)\right\} \qquad [A.13]$$

First values of l (symbol) and m (frame) are calculated for every "t" instant and the signal is created using the cmlk matrix. Then the modulation in frequency and the phy function are added for every "t" instant. Finally the vector with the modulated ST is returned.

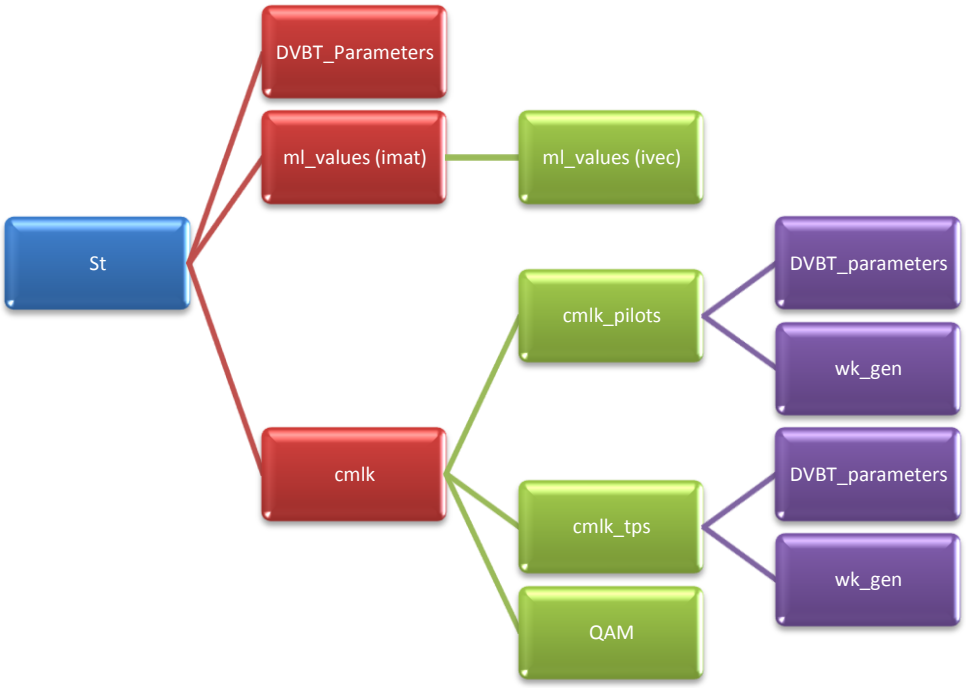

Figure A.5.St scheme.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Ivec	DVB	DVB(0) = BW — Bandwidth 6, 7, 8 (MHz) DVB(1) = Mode — Mode of operation 2 (2k mode) or 8 (8k mode) DVB(2) = GI — Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)
cmat	cmlk	Matrix of 68·K with all the pilots inserted (scattered, continual and TPS) and modulated data added.

vec	t	Vector of "t" samples.
double	fc	Central frequency to modulate the signal.
OUTPUTS		
Type	**Variable**	**Description**
vec	loc_vec	Returns a real vector with the ST signal for the given number of "t" samples

Table A.9 Input & Output st

ts

Description:

Calculate a vector of samples of determinate length. The distance between samples is TS. The vector with all the "t" samples is returned.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Int	length	Number of samples.
double	start	Value for the first "t" sample
double	Ts	Distance between one sample and the consecutive one.
OUTPUTS		
Type	**Variable**	**Description**
vec	loc_vec	Returns a vector of "t" samples.

Table A.10 Input & Output ts

St0

Description:

Generate a vector with the ST signal for the direct path for a time instant (not the matrix, only for K carriers). The ck vector is modulated on frequency and multiplied by the phy function.

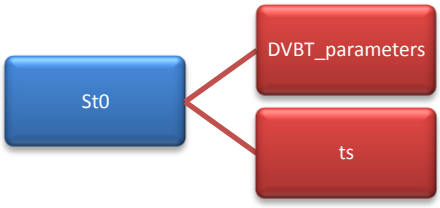

Figure A.6.St0 scheme.

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Ivec	DVB	DVB(0) = BW Bandwidth 6, 7, 8 (MHz) DVB(1) = Mode Mode of operation 2 (2k mode) or 8 (8k mode) DVB(2) = GI Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)
Cvec	ck	Vector with all the carriers for an instant time
Vec	q	Delay of the path
double	fc	Central frequency.
OUTPUTS		
Type	**Variable**	**Description**
Cvec	loc_cvec	Returns a vector with all the carriers with the ST signal for the direct path

Table A.11 Input & Output st0

SG_DVBT

Description:

Generate a complex vector with the generation of the DVB-T signal after crossing the channel transfer function (multipath). This channel can be Rayleigh, Rician or AGWN. The expression of s(t) and of the channel is showed below.

$$s(t) = Re\left\{e^{j2\pi f_c t}\sum_{m=0}^{\infty}\sum_{l=0}^{67}\sum_{k=k_{min}}^{k_{max}} c_{m,l,k}\cdot\varphi_{m,l,k}(t)\right\} \qquad [A.14]$$

- Fix Reception Rician:

$$y(t) = \frac{\rho_0 x(t)+\sum_{i=1}^{N}\rho_i e^{-j\theta_i}x(t-\tau_i)}{\sqrt{\sum_{i=1}^{N}\rho_i{}^2}} \qquad [A.15]$$

- Portable reception Rayleigh

$$y(t) = \frac{\sum_{i=1}^{N} \rho_i e^{-j\theta_i} x(t-\tau_i)}{\sqrt{\sum_{i=1}^{N} \rho_i{}^2}} \qquad [A.16]$$

First the cmlk matrix is created. Depending on "seed" the data introduced is random or a given vector. Then a direct path is created (St0) and afterwards the rest of the paths are formed. Finally the complete signal is returned.

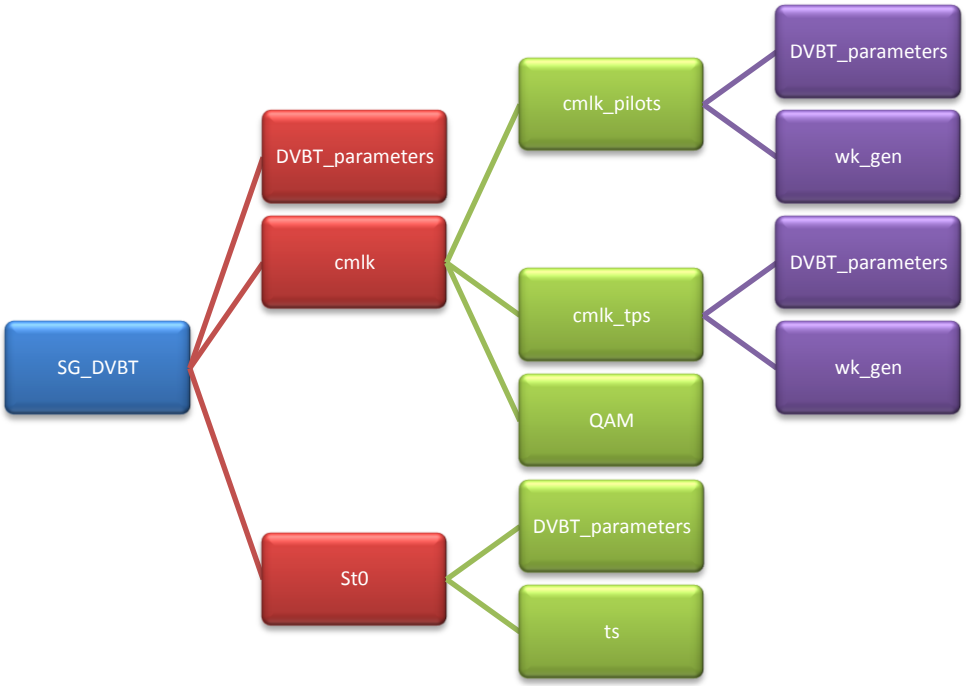

Figure A.7. SG_DVBT scheme

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
Ivec	DVB	DVB(0) = BW — Bandwidth 6, 7, 8 (MHz) DVB(1) = Mode — Mode of operation 2 (2k mode) or 8 (8k mode) DVB(2) = GI — Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI)
ivec	Modp	modp(0) = Level — Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6 2 for QPSK (2 bits) 4 for 16 QAM (4 bits) 6 for 64 QAM (6 bits)

		modp(1) = alpha Proportions of the constellation. 0 (non-hierarchical) 1, 2 or 4 (hierarchical)
Vec	Codep	Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then cr2 = 0. codep(0) = cr1 codep(1) = cr2
`cvec	H	Modeled channel
Vec	Tau	Values of the time delay parameters
Double	T	Duration of the simulation
double	Fc	Central frequency.
int	Seed	The seed for random data generator. If this value is 0 then the seed is randomized by setting. It's value to sum (100·clock).
int	K	It is the oversampling factor. The DVB-T signal is usually sampled at the rate defined by the DVB-T standard (K=1). If K is an integer greater than 1, then the values of the signal is uniformly placed (K=1) points within each sampling period are also given.

OUTPUTS		
Type	**Variable**	**Description**
cvec	loc_cvec	Return a complex vector with the generation signal given, for a T time.

Table A.12 Input & Output SG_DVBT

AC_detector

Description:

Model the behavior of the AC_detector. The statistic of this detection algorithm is showed below:

$$\Lambda_{AC} = \frac{1}{2(M+T_D)} \sum_{t=0}^{M+T_D-1} x^2(t) \qquad [A.17]$$

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
cvec	X	The received signal

int	Td	Number of carriers (N)
OUTPUTS		
Type	**Variable**	**Description**
double	stemp	Returns the statistics of the detector

Table A.13 Input & Output AC_detector

CP_detector

Description:

Model the behavior of the CP_detector. The statistic of this detection algorithm is showed below:

$$\Lambda_{CP} = \left|\sum_{k=\theta}^{\theta+L-1} r^*(k)r(k+N)\right| \quad \text{[A.18]}$$

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
cvec	x	The received signal
int	CP	Number of carriers of the CP (Np)
int	Td	Number of carriers (N)
Int	out	Select the real (0) or the absolute value (1) of the statistic
OUTPUTS		
Type	**Variable**	**Description**
double	stemp	Returns the statistics of the detector

Table A.14 Input & Output CP_detector

Pilot_detector

Description:

Model the behavior of the Pilot_detector. The statistic of this detection algorithm is showed below:

$$\Lambda_{PI} = \left|\sum_{k=\theta}^{\theta+N+L-1} r^*(k)m(k-\theta)\right| \quad \text{[A.19]}$$

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
cvec	x	The received signal
cvec	m	The pilot signal
Int	out	Select the real (0) or the absolute value (1) of the statistic
OUTPUTS		
Type	**Variable**	**Description**
double	stemp	Returns the statistics of the detector

Table A.15 Input & Output Pilot_detector

CP_Pilot_detector

Description:

Model the behavior of the CP_Pilot_detector. The statistic of this detection algorithm is showed below:

$$\Lambda_{CP\_PI} = \left|\sum_{k=\theta}^{\theta+L-1}(r(k)+r(k+N))^{*}m(k-\theta)\right| \qquad \text{[A.20]}$$

Inputs & Outputs:

INPUTS		
Type	**Variable**	**Description**
cvec	x	The received signal
cvec	m	The pilot signal
int	CP	Number of carriers of the CP (Np)
int	Td	Number of carriers (N)
Int	out	Select the real (0) or the absolute value (1) of the statistic
OUTPUTS		
Type	**Variable**	**Description**
double	stemp	Returns the statistics of the detector

Table A.16 Input & Output CP_Pilot_detector

Appendix B

C++ Functions

All the functions created are shown below:

```
void Fading_Parameters(string type, cvec& hl, vec& taul) {
    /* Model the behavior of the channel. Multipath channel with 20 paths.
    * type: define the kind of channel to simulate “Rayleigh, Rician or AWGN.
    * hl: model the channel.
    * taul: values of the time delay parameters. */

    // power delay
    vec rho = "0.057662 0.176809 0.407163 0.303585 0.258782 0.061831 0.150340 0.051534 0.185074
    0.400967 0.295723 0.350825 0.262909 0.225894 0.170996 0.149723 0.240140 0.116587 0.221155
    0.259730";

    // phase rotation
    vec theta = "4.855121 3.419109 5.864470 2.215894 3.758058 5.430202 3.952093 1.093586 5.775198
    0.154459 5.928383 3.053023 0.628578 2.128544 1.099463 3.462951 3.664773 2.833799 3.334290
    0.393889";

    // time delay
    taul = "1.003019 5.422091 0.518650 2.751772 0.602895 1.016585 0.143556 0.153832 3.324866 1.935570
    0.429948 3.228872 0.848831 0.073883 0.203952 0.194207 0.924450 1.381320 0.640512 1.368671"; taul *=
    pow(10, -6);

    if (type == "rician") {
        // The LOS path if we have Rician fading. Rician factor K(in db), Klin(linear)
        double K = 10.0;
        double Klin = pow(10, K / 10);
        double rho0 = sqrt(Klin * rho * rho); // amplification(no phase and time shift)
        rho.ins(0, rho0);
        theta.ins(0, 0);
        taul.ins(0, 0);
        rho = rho / (sqrt(rho * rho));
    }

    if (type == "rayleigh") {
        rho = rho / (sqrt(rho * rho));
    }

    if (type == "AWGN") {
        rho = "1";
        theta = "0";
        taul = "0";
    }

    // Calculation of the complex multiplicative factor hl
    hl.set_length(theta.length());

    for (int k = 0; k < theta.length(); k++) {
        hl(k) = polar(1.0, -theta(k));
    }
```

```
    elem_mult_inplace(to_cvec(rho), hl);
}

void DVBT_parameters(ivec DVB, vec &Times, ivec & Pars) {
    /* Obtaining all the parameters needed to generate the emitted signal
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
     * Times(0) = T =>      Sampling interval in seconds
     * Times(1) = Tu =>     Useful part duration
     * Times(2) = Delta =>  Guard interval duration
     * Times(3) = TS =>     Total symbol duration
     * Pars(0) = Kmin =>    First carrier number
     * Pars(1) = Kmax =>    Last carrier number
     * Pars(2) = K =>       Number of carriers per symbol
     * Pars(3) = N =>       Number of samples per symbol */

    Times.set_size(4);
    Pars.set_size(4);
    Times(0) = 7e-6 / (8 * DVB(0));                 // The sampling interval in seconds
    Times(1) = 1024 * DVB(1) * Times(0);            // Useful part duration
    Times(2) = Times(1) * pow(2, -DVB(2));          // Guard interval duration
    Times(3) = Times(1) + Times(2);                 // Total symbol duration
    Pars(0) = 0;                                    // First carrier number
    Pars(1) = 852 * DVB(1);                         // Last carrier number
    Pars(2) = Pars(1) - Pars(0) + 1;                // Number of Carriers
    Pars(3) = (1024 + pow(2, 10 - DVB(2))) * DVB(1); // Number of samples per symbol
}

cvec QAM(bvec InVec, ivec modp) {
    /* Generation of a vector of coefficients (complex value) of a constellation from a binary sequence which
    are the information data
     * InVec = Binary sequence.
     * modp(0) = level => Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6.
     *              - 2 for QPSK (2 bits)
     *              - 4 for 16 QAM (4 bits)
     *              - 6 for 64 QAM (6 bits)
     * modp(1) = Alpha => Minimum distance separating two constellation points.
     *              - 0 (non-hierararchical)
     *              - 1, 2 or 4 (hierararchical)
     * NF = Normalization Factor per E[c x c*] = 1.
     *      - QPSK = 2
     *      - 16 QAM 1 (alpha) = 10
     *      - 16 QAM 2 (alpha) = 20
     *      - 16 QAM 4 (alpha) = 52
     *      - 64 QAM 1 (alpha) = 42
     *      - 64 QAM 2 (alpha) = 60
     *      - 64 QAM 4 (alpha) = 108
     * nr_symbols = Number of symbols of the Binary sequence (InVec).
     * map = complex vector with the values of the constellation.
     * Returning cvec vector of nr_symbols dimension, with all mapping data */

    cvec loc_vec, map;
```

```
int level = modp(0);
int alpha = modp(1);
int NF = 1, nr_symbols = InVec.length() / level;

it_assert(mod(InVec.length(), level) == 0, "Wrong length of binary sequence, (has to be divisible by nr of
levels)");
it_assert(level == 2 || level == 4 || level == 6, "QAM level, out of range (has to be in {2,4,6})");
it_assert(alpha == 0 || alpha == 1 || alpha == 2 || alpha == 4, "alpha, out of range (has to be in {0,1,2,4})");

switch (alpha) {
    case 0: case 1:
        switch (level) {
            case 2:
                map = "1+1i 1-1i -1+1i -1-1i";
                NF = 2;
                break;
            case 4:
                map = "3+3i 3+1i 1+3i 1+1i 3-3i 3-1i 1-3i 1-1i -3+3i -3+1i -1+3i -1+1i -3-3i -3-1i -1-31i -1-1i";
                NF = 10;
                break;
            case 6:
                map = "7+7i 7+5i 5+7i 5+5i 7+1i 7+3i 5+1i 5+3i 1+7i 1+5i 3+7i 3+5i 1+1i 1+3i 3+1i 3+3i 7-7i 7-5i
                5-7i 5-5i 7-1i 7-3i 5-1i 5-3i 1-7i 1-5i 3-7i 3-5i 1-1i 1-3i 3-1i 3-3i -7+7i -7+5i -5+7i -5+5i -7+1i -
                7+3i -5+1i -5+3i -1+7i -1+5i -3+7i -3+5i -1+1i -1+3i -3+1i -3+3i -7-7i -7-5i -5-7i -5-5i -7-i -7-3i -5-
                1i -5-3i -1-7i -1-5i -3-7i -3-5i -1-1i -1-3i -3-1i -3-3i";
                NF = 42;
            default:
                break;
        }
        break;
    case 2:
        switch (level) {
            case 4:
                map = "4+4i 4+2i 2+4i 2+2i 4-4i 4-2i 2-4i 2-2i -4+4i -4+2i -2+4i -2+2i -4-4i -4-2i -2-4i -2-2i";
                NF = 20;
                break;
            case 6:
                map = "8+8i 8+6i 6+8i 6+6i 8+2i 8+4i 6+2i 6+4i 2+8i 2+6i 4+8i 4+6i 2+2i 2+4i 4+2i 4+4i 8-8i 8-6i
                6-8i 6-6i 8-2i 8-4i 6-2i 6-4i 2-8i 2-6i 4-8i 4-6i 2-2i 2-4i 4-2i 4-4i -8+8i -8+6i -6+8i -6+6i -8+2i -
                8+4i -6+2i -6+4i -2+8i -2+6i -4+8i -4+6i -2+2i -2+4i -4+2i -4+4i -8-8i -8-6i -6-8i -6-6i -8-2i -8-4i -
                6-2i -6-4i -2-8i -2-6i -4-8i -4-6i -2-2i -2-4i -4-2i -4-4i";
                NF = 60;
            default:
                break;
        }
        break;
    case 4:
        switch (level) {
            case 4:
                map = "6+6i 6+4i 4+6i 4+4i 6-6i 6-4i 4-6i 4-4i -6+6i -6+4i -4+6i -4+4i -6-6i -6-4i -4-6i -4-4i";
                NF = 52;
                break;
            case 6:
                map = "10+10i 10+8i 8+10i 8+8i 10+4i 10+6i 8+4i 8+6i 4+10i 4+8i 6+10i 6+8i 4+4i 4+6i 6+4i
                6+6i 10-10i 10-8i 8-10i 8-8i 10-4i 10-6i 8-4i 8-6i 4-10i 4-8i 6-10i 6-8i 4-4i 4-6i 6-4i 6-6i -10+10i -
```

```
                    10+8i -8+10i -8+8i -10+4i -10+6i -8+4i -8+6i -4+10i -4+8i -6+10i -6+8i -4+4i -4+6i -6+4i -6+6i -
                    10-10i -10-8i -8-10i -8-8i -10-4i -10-6i -8-4i -8-6i -4-10i -4-8i -6-10i -6-8i -4-4i -4-6i -6-4i -6-6i";
                    NF = 108;
                    break;
                default:
                    break;
            }
            break;
        default:
            break;
    }

    loc_vec.set_length(nr_symbols);

    for (int k = 0; k < nr_symbols; k++) {
        loc_vec(k) = map(bin2dec(InVec.get(k*level, (k + 1) * level - 1))) / sqrt(double(NF));
    }
    return loc_vec;
}
```

```
cvec wk_gen(int nr_carriers) {
    /* Generation of the reference sequence from the PRBS sequence (wk) modulated for scattered and
    continual pilots.
    * nr_carriers = Number of carriers for wk. */
    LFSR wk_reg;
    bvec cons = "1 0 0 0 0 0 0 0 0 1 0 1";          // Generator polynomial.
    bvec istate = "1 1 1 1 1 1 1 1 1 1 1";          // Initial seed.
    cvec loc_vec;

    loc_vec.set_length(nr_carriers);
    wk_reg.set_connections(cons);
    wk_reg.set_state(istate);

    for (int k = 0; k < 11; k++) {
        loc_vec(k) = 4.0 * (1 - 2 * double(istate(k))) / 3;      // Modulation of the first 11 carriers.
    }

    for (int k = 11; k < nr_carriers; k++) {
        loc_vec(k) = 4.0 * (1 - 2 * double(wk_reg.shift())) / 3;  // Modulation of the rest of the carriers.
    }
    return loc_vec;
}
```

```
ivec ml_values(double Ts, double t) {
    /* Generate a vector with the values of l (symbol) and m (frame) for a given “t”
    * Ts = Symbol time.
    * t = Instant of time to calculate the values of l and m.
    */
    ivec loc_vec;

    loc_vec.set_length(2);
    loc_vec(1) = int(t / Ts);
    loc_vec(0) = loc_vec(1) / 68;              // m value.
```

```
    loc_vec(1) = mod(loc_vec(1), 68);          // l value.
    return loc_vec;
}


imat ml_values(double Ts, vec t) {
    /* Generate a vector with the values of l (symbol) and m (frame) for a vector of given "t"
    * Ts = Symbol time.
    * t = Vector of times to calculate the values of l and m.
    */

    imat loc_imat;

    loc_imat.set_size(2, t.length(), false);

    for (int i = 0; i < t.length(); i++){
        loc_imat.set_col(i, ml_values(Ts, t(i)));
    }
    return loc_imat;
}


cmat cmlk_pilots(ivec DVB) {
    /* Generate a matrix with the continual and scattered pilots
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
    */

    vec times;
    ivec iparams, pos;
    cvec wk;
    cmat loc_cmat;
    const int LC = 1704;
    int lm;

    DVBT_parameters(DVB, times, iparams);
    loc_cmat.set_size(68, iparams(2), false);
    loc_cmat.zeros();
    wk = wk_gen(iparams(2));
    // Position of the continual pilot
    pos = "48 54 87 141 156 192 201 255 279 282 333 432 450 483 525 531 618 636 714 759 765 780 804 873
    888 918 939 942 969 984 1050 1101 1107 1110 1137 1140 1146 1206 1269 1323 1377 1491 1683 1704";

    // Continual pilots
    for (int l = 0; l < 68; l++) {
        loc_cmat(l, 0) = wk(0);
        for (int s = 0; s < DVB(1) / 2; s++) {
            for (int k = 0; k < pos.length(); k++) {
                loc_cmat(l, pos(k) + s * LC) = wk(pos(k) + s * LC);
            }
        }
    }
```

```
    // Scattered pilots
    for (int l = 0; l < 68; l++) {                    // Calculate the position of the scattered pilots.
        lm = 3 * mod(l, 4);
        for (int s = 0; s < double(iparams(2) - lm) / 12; s++) {
            loc_cmat(l, lm + 12 * s) = wk(lm + 12 * s);
        }
    }
    return loc_cmat;
}


cmat cmlk_tps(int m, ivec DVB, ivec modp, vec codep) {
    /* Generate a matrix with the tps pilots
     * m =  Frame in the super frame. 1,2,3 or 4
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
     * modp(0) = level => Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6.
     *                - 2 for QPSK (2 bits)
     *                - 4 for 16 QAM (4 bits)
     *                - 6 for 64 QAM (6 bits)
     * modp(1) = Alpha => Minimum distance separating two constellation points.
     *                - 0 (non-hierararchical)
     *                - 1, 2 or 4 (hierararchical)
     * codep =  Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then
    cr2 = 0.
     * codep(0) = cr1
     * codep(1) = cr2 */

    vec times;
    ivec iparams, pos;
    cvec wk;
    cmat loc_cmat;
    bvec s;
    int fn;
    const int LC = 1704;

    DVBT_parameters(DVB, times, iparams);
    loc_cmat.set_size(68, iparams(2), false);
    loc_cmat.zeros();
    wk = wk_gen(iparams(2));
    pos = "34 50 209 346 413 569 595 688 790 901 1073 1219 1262 1286 1469 1594 1687";  //position of the
    tps pilots
    s.set_size(67);
    s.zeros();
    fn = mod(m, 4) + 1;

    // Synchronization word
    if ((fn == 1) || (fn == 3))
        s.set_subvector(0, 15, "0 0 1 1 0 1 0 1 1 1 1 0 1 1 1 0");
    else
        s.set_subvector(0, 15, "1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 1");

    // Length indicator
    s.set_subvector(16, 21, "0 1 0 1 1 1");
```

```
// Frame number
switch (fn) {
    case 2:
        s.set_subvector(22, 23, "0 1");
        break;
    case 3:
        s.set_subvector(22, 23, "1 0");
        break;
    case 4:
        s.set_subvector(22, 23, "1 1");
        break;
    default:
        break;
}

// QAM levels
switch (modp(0)) {
    case 4:
        s.set_subvector(24, 25, "0 1");
        break;
    case 6:
        s.set_subvector(24, 25, "1 0");
        break;
    default:
        break;
}

// Hierarchy information (alpha)
switch (modp(1)) {
    case 1:
        s.set_subvector(26, 28, "0 0 1");
        break;
    case 2:
        s.set_subvector(26, 28, "0 1 0");
        break;
    case 4:
        s.set_subvector(26, 28, "0 1 1");
        break;
    default:
        break;
}

// HP stream code rate
if (codep(0) == 2.0 / 3.0)
    s.set_subvector(29, 31, "0 0 1");
if (codep(0) == 3.0 / 4.0)
    s.set_subvector(29, 31, "0 1 0");
if (codep(0) == 5.0 / 6.0)
    s.set_subvector(29, 31, "0 1 1");
if (codep(0) == 7.0 / 8.0)
    s.set_subvector(29, 31, "1 0 0");

// LP stream code rate
if (codep(1) == 2.0 / 3.0)
    s.set_subvector(32, 34, "0 0 1");
```

```
if (codep(1) == 3.0 / 4.0)
    s.set_subvector(32, 34, "0 1 0");
if (codep(1) == 5.0 / 6.0)
    s.set_subvector(32, 34, "0 1 1");
if (codep(1) == 7.0 / 8.0)
    s.set_subvector(32, 34, "1 0 0");

// Guard interval(GI)
switch (DVB(2)) {
    case 2:
        s.set_subvector(35, 36, "1 1");
        break;
    case 3:
        s.set_subvector(35, 36, "1 0");
        break;
    case 4:
        s.set_subvector(35, 36, "0 1");
        break;
    default:
        break;
}

// Transmission mode (mode)
if (DVB(1) == 8)
    s.set_subvector(37, 38, "0 1");

// Bits 39 - 46 Reserved for cell_id information ---> zeros
// Bits 47 - 52 Reserved for future use ---> zeros
// Error protection of the TPS pilots -> Shortened binary BCH (67, 53, 5) - code

bvec gx = "1 1 1 0 1 1 1 0 1 1 0 0 0 1";  // The generator polynomial for the BCH code.
bvec LFSRstate = s.mid(39, 14);
bvec inbits;
inbits.set_size(53);
inbits.zeros();
inbits.set_subvector(14, 52, s.left(39));

for (int i = 52; i > -1; i--) {
    if (LFSRstate(13)) {
        LFSRstate.set_subvector(1, 13, LFSRstate.left(13));
        LFSRstate(0) = inbits(i);
        LFSRstate += gx.left(14);
    }
    else {
        LFSRstate.set_subvector(1, 13, LFSRstate.left(13));
        LFSRstate(0) = inbits(i);
    }
}
s.set_subvector(53, 66, LFSRstate);

// The TPS pilots
for (int r = 0; r < DVB(1) / 2; r++) {
    for (int k = 0; k < pos.length(); k++) {
        loc_cmat(0, pos(k) + r * LC) = 3 * wk(pos(k) + r * LC) / 4;
    }
```

```
    }

    for (int l = 0; l < 67; l++) {
        for (int r = 0; r < DVB(1) / 2; r++) {
            for (int k = 0; k < pos.length(); k++) {
                if (s(l)) {
                    loc_cmat(l + 1, pos(k) + r * LC) = -loc_cmat(l, pos(k) + r * LC);
                }
                else {
                    loc_cmat(l + 1, pos(k) + r * LC) = loc_cmat(l, pos(k) + r * LC);
                }
            }
        }
    }
    return loc_cmat;
}
```

```
cmat cmlk(int m, ivec DVB, ivec modp, vec codep) {
    /* Generate a matrix with all the pilots (continual, scattered and TPS pilots) and modulated random data.
     * m =  Frame in the super frame. 1,2,3 or 4
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
     * modp(0) = level => Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6.
     *                - 2 for QPSK (2 bits)
     *                - 4 for 16 QAM (4 bits)
     *                - 6 for 64 QAM (6 bits)
     * modp(1) = Alpha => Minimum distance separating two constellation points.
     *                - 0 (non-hierarchical)
     *                - 1, 2 or 4 (hierarchical)
     * codep =  Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then
    cr2 = 0.
     * codep(0) = cr1
     * codep(1) = cr2 */

    cmat loc_cmat;
    bvec loc_bvec;
    cvec loc_cvec;
    const int PC = 1512;
    int counter = 0;

    loc_cmat = cmlk_pilots(DVB);
    loc_cmat += cmlk_tps(m, DVB, modp, codep);

    RNG_randomize();
    loc_bvec = Bernoulli_RNG()(68 * PC * modp(0) * DVB(1) / 2);
    loc_cvec = QAM(loc_bvec, modp);

    for (int l = 0; l < 68; l++) {
        for (int k = 0; k < loc_cmat.cols(); k++) {
            if (abs(loc_cmat(l, k)) < 0.01) {
                loc_cmat(l, k) = loc_cvec(counter);
                counter++;
            }
```

```
        }
    }
    return loc_cmat;
}
```

```
cmat cmlk(int m, ivec DVB, ivec modp, vec codep, bvec data) {
    /* Generate a matrix with all the pilots (continual, scattered and TPS pilots) and modulated data.
     * m =  Frame in the super frame. 1,2,3 or 4
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
     * modp(0) = level => Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6.
     *                - 2 for QPSK (2 bits)
     *                - 4 for 16 QAM (4 bits)
     *                - 6 for 64 QAM (6 bits)
     * modp(1) = Alpha => Minimum distance separating two constellation points.
     *                - 0 (non-hierararchical)
     *                - 1, 2 or 4 (hierararchical)
     * codep =  Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then
    cr2 = 0.
     * codep(0) = cr1
     * codep(1) = cr2
     * data = values for the data (in this case is not random)*/

    cmat loc_cmat;
    cvec loc_cvec;
    const int PC = 1512;
    int counter = 0;

    if (data.length() == 1) {
        bvec ldata;
        ldata.set_length(68 * PC * modp(0) * DVB(1) / 2);

        // set the data vector to all-zero or all-one vector if the input data is "0" or "1", respectively
        if (data(0))
            ldata.ones();
        else
            ldata.zeros();
        loc_cvec = QAM(ldata, modp);
    }
    else {
        it_assert(data.length() == 68 * PC * modp(0) * DVB(1) / 2, "The binary data vector has no proper
        length!");
        loc_cvec = QAM(data, modp);
    }

    loc_cmat = cmlk_pilots(DVB);                // adding continual and scattered pilots
    loc_cmat += cmlk_tps(m, DVB, modp, codep);     // adding TPS pilots

    for (int l = 0; l < 68; l++) {
        for (int k = 0; k < loc_cmat.cols(); k++) {
            if (abs(loc_cmat(l, k)) < 0.01) {

                loc_cmat(l, k) = loc_cvec(counter);
```

```
                counter++;
            }
        }
    }
    return loc_cmat;
}
```

```
vec st(ivec DVB, cmat cmlk, vec t, double fc) {
    /* Generate a a vector with the ST signal for a given number of samples of "t".
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
     * Matrix of 68·K with all the pilots inserted (scattered, continual and TPS) and modulated data added.
     * t = Vector of times.
     * fc = Central frequency to modulate the signal. */

    double Ts, Tu, delta;
    int Kmin, Kmax, m, l, nr_samples = t.length();
    complex<double> cptemp(0, 0), ctemp(0, 0), cft(0, 0);
    vec loc_vec, times;
    ivec iparams, ml;

    DVBT_parameters(DVB, times, iparams);
    Tu = times(1);
    delta = times(2);
    Ts = times(3);
    Kmin = iparams(0);
    Kmax = iparams(1);
    loc_vec.set_size(nr_samples);

    for (int j = 0; j < nr_samples; j++) {
        ml = ml_values(Ts, t(j));                   // extracting the l (symbol) m (frame) values
        m = ml(0);
        l = ml(1);
        cptemp.imag() = 2 * PI * fc * t(j);
        cft = exp(cptemp);                          // expression to modulate on frequency
        ctemp = 0;
        for (int k = Kmin; k < Kmax; k++) {         // creating the phi function
            cptemp.imag() = 2 * PI * double(k - (Kmin + Kmax) / 2)*(t(j) - delta - (l + 68 * m) * Ts) / Tu;
            ctemp += cmlk(l, k) * exp(cptemp);
        }
        ctemp *= cft;
        loc_vec(j) = ctemp.real();                  // only the real part is important
    }
    return loc_vec;
}
```

```
vec ts(int length, double start, double ts) {
    /* Calculate a vector of samples of determinate length
     * length = number of samples.
     * start = value for the first "t" sample.
     * ts = distance between one sample and the next one. */
    vec loc_vec;
```

```
    loc_vec.set_length(length);
    loc_vec(0) = start;

    for (int i = 1; i < length; i++) {
        loc_vec(i) = loc_vec(i - 1) + ts;
    }
    return loc_vec;
}


cvec modulate(cmat inmat, int Nfft, int Ncp, double t0tots) {
    OFDM mod;
    cvec tcv, ph, loc_cvec = "";
    int nr_carriers = inmat.cols();

    ph.set_length(nr_carriers);

    for (int k = 0; k < nr_carriers; k++) {
        ph(k) = polar(1.0, 2 * PI * t0tots * (k - (nr_carriers - 1) / 2) / Nfft);
    }

    mod.set_parameters(Nfft, Ncp, 1);
    for (int i = 0; i < inmat.rows(); i++) {
        tcv = inmat.get_row(i);
        tcv = elem_mult(tcv, ph);
        tcv = concat(tcv.right(1 + nr_carriers / 2), zeros_c(Nfft - nr_carriers), tcv.left(nr_carriers / 2));
        loc_cvec = concat(loc_cvec, mod.modulate(tcv));
    }
    return loc_cvec;
}

cvec st0(ivec DVB, cvec ck, double q, double fc) {
    /* generate a vector with the ST signal for the direct path for a time instant (not the matrix, only for K
    carriers)
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>    Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).
     * ck = Vector with all the carrier for an instant time
     * q = delay of the path
     * fc - the carrier frequency fc in Hz */

    ivec Pars;
    vec Times, t;
    cvec ph, loc_cvec;
    OFDM ofdm;

    DVBT_parameters(DVB, Times, Pars);
    double ts_loc = Times(0);
    int Kmax = Pars(1);
    double t0 = q*ts_loc;
    it_assert(q >= 0 && q < 1, "q, out of range (has to be in the interval [0,1))!");
    it_assert(ck.length() == Kmax + 1, "The coefficients vector has wrong length!");
    int Nfft = 1024 * DVB(1);
    int Ncp = pow(2, 10 - DVB(2)) * DVB(1);
    ofdm.set_parameters(Nfft, Ncp, 1);
```

```
    ph.set_length(ck.length());

    for (int k = 0; k < ph.length(); k++) {
        ph(k) = polar(1.0, 2 * PI * q * (k - Kmax / 2) / Nfft);                    // phi function
    }

    loc_cvec = elem_mult(ck, ph);
    loc_cvec = ofdm.modulate(concat(loc_cvec.right(1 + Kmax / 2), zeros_c(Nfft - Kmax - 1), loc_cvec.left(Kmax
    / 2)));
    t = ts(Pars(3), t0, ts_loc);

    for (int k = 0; k < loc_cvec.length(); k++) {
        loc_cvec(k) *= polar(1.0, 2 * PI * fc * t(k));                    // frequency modulation
    }
    return loc_cvec;
}
```

cvec SG_DVBT(ivec DVB, ivec modp, vec codep, cvec h, vec tau, double T, double fc, int seed, int K)

```
{
    /* Generate a complex vector with the generation of the DVB-T signal after crossing the channel transfer
    function (multipath).
     * DVB(0) = BW =>       Bandwidth 6,7,8 (MHz)
     * DVB(1) = mode =>     Mode of operation 2 (2k mode) or 8 (8k mode)
     * DVB(2) = GI =>       Guard interval 2, 3, 4 or 5 corresponding to 2^(-GI).

    * modp(0) = level => Modulation used. Number of bits in the sequence. Column vector with 2, 4 or 6.
     *                - 2 for QPSK (2 bits)
     *                - 4 for 16 QAM (4 bits)
     *                - 6 for 64 QAM (6 bits)
     * modp(1) = Alpha => Minimum distance separating two constellation points.
     *                - 0 (non-hierararchical)
     *                - 1, 2 or 4 (hierararchical)
    * codep =  Inner code rates, cr1 (channel1) & cr2 (channel2). 0, 1/2, 2/3, 3/4, 5/6 or 7/8. If alpha = 0, then
    cr2 = 0.
     *       codep(0) = cr1
     *       codep(1) = cr2 */
     * h = vector [h0, h1,...,hk] with complex path amplification coefficients
     * tau - the vector [tau_0,tau_1,...,tau_k] with delays for the different paths
     * T - the duration of the signal T
     * fc - the carrier frequency fc in Hz
     * seed - the seed for the random data generator. If this value is 0 then the seed is randomized by setting
    it's value to sum(100*clock).
     * K - the oversamplings factor K. The DVB-T signal is usually sampled at the rate defined by the DVB-T
    standard (K=1). If K is an integer greater than 1 then the values of the signal in uniformly placed (K-1) points
    within each sampling period are also given. */

    ivec Pars;
    vec Times;
    vec curr_tau;
    ivec delay_ind;
    int curr_ind, curr_m, curr_l, a;
    vec q;
    cvec sym;
```

```
    DVBT_parameters(DVB, Times, Pars);
    int N = ceil(T / Times(0));                    // number of samples
    int M = ceil(double(N) / (68 * Pars(3)));   // number of frames
    cmat C[M];

    // create the matrix cmlk with m frame. Cmlk is only calculated for one frame. Depending on seed, data is
    random or not.
    for (int i = 0; i < M; i++) {
        if (seed)
            C[i] = cmlk(i, DVB, modp, codep, "0");
        else
            C[i] = cmlk(i, DVB, modp, codep);
    }

    cvec signal = zeros_c(K * N);

    for (int i = 0; i < K; i++) {
        curr_tau = (i * Times(0) / K) + tau;
        delay_ind = 1 + ceil_i(curr_tau / Times(0));
        q = delay_ind - 1 - curr_tau / Times(0);
        for (int path = 0; path < tau.length(); path++) {     // create the different path for the multipath channel
            curr_ind = delay_ind(path);
            curr_m = 0;
            curr_l = 0;
            while (curr_ind < N) {
                sym = h(path) * st0(DVB, C[curr_m].get_row(curr_l), q(path), fc);
                a = curr_ind;
                while (((a * K - i - 1) < signal.length()) && (a < curr_ind + Pars(3))) {
                    signal(a * K - i - 1) += sym(a - curr_ind);
                    a++;
                }
                curr_l++;
                curr_ind += Pars(3);
                if (curr_l == 68) {
                    curr_l = 0;
                    curr_m++;
                }
            }
        }
    }

    if (K > 1)
        signal.del(0, K - 2);
    return signal;
}


double Pilot_detector(cvec x, cvec m, int out) {
    /* Model the behavior of the Pilot_detector
     * x = received signal.
     * m = pilot signal.
     * out = Select the real (0) or the absolute value (1) of the statistic
    */

    complex<double> stemp(0, 0);
```

```
    for (int i = 0; i < x.length() / m.length(); i++) {
        stemp += x.mid(i * m.length(), m.length()) * conj(m);
    }
    switch (out){
        case 0:
            return real(stemp);
            break;
        case 1:
            return abs(stemp);
            break;
    }
}
```

```
double CP_detector(cvec x, int CP, int Td, int out) {
    /* Model the behavior of the CP_detector
     * x = received signal.
     * CP = number of carriers of the CP (Np)
     * Td = number of carriers (N)
     * out = Select the real (0) or the absolute value (1) of the statistic
    */

    cvec xl = x.left(x.length() - Td);
    cvec xr = x.right(x.length() - Td);
    int i = 0, j = 0;
    complex<double> stemp(0, 0);
    int k = 0, n = 0;

    while (k < xl.length()) {
        stemp += xl(k) * conj(xr(k));
        j++;
        n++;
        if (j == CP) {
            i++;
            j = 0;
        }
        k = (Td + CP) * i + j;
    }
    switch (out){
        case 0:
            return real(stemp);
            break;
        case 1:
            return abs(stemp);
            break;
    }
}
```

```
double CP_pilot_detector(cvec x, cvec m, int CP, int Td, int out) {
    /* Model the behavior of the CP_pilot_detector
     * x = received signal.
     * m = pilot signal.
     * CP = number of carriers of the CP (Np)
     * Td = number of carriers (N)
```

```
     * out = Select the real (0) or the absolute value (1) of the statistic
    */

    cvec xl = x.left(x.length() - Td);
    cvec xr = x.right(x.length() - Td);
    int i = 0, j = 0;
    complex<double> stemp(0, 0);
    int k = 0, n = 0;

    while (k < xl.length()) {
        stemp += (xl(k) + xr(k)) * conj(m((Td + CP) * (i * 4) + j));
        j++;
        n++;
        if (j == CP) {
          i++;
          j = 0;
        }
        k = (Td + CP) * i + j;
    }
    switch (out){
        case 0:
            return real(stemp);
            break;
        case 1:
            return abs(stemp);
            break;
    }
}


inline double Qinv(double x) {
    return sqrt(2) * erfinv(1 - 2 * x);
};


inline double cnorm(cvec invec) {
    return sqrt((invec * conj(invec)).real());
};


inline double Power(cvec x, double ts) {
    return (x * conj(x)).real() / (x.length() * ts);
};


inline double Power_Th(double Pfa, double sigma, int M, double ts) {
    return sigma * (1 + Qinv(Pfa) / sqrt(M)) / ts;
};


inline double Power_Pmd(double Pfa, double P, double sigma, int M, double ts) {
    return Qfunc((sqrt(M) * P - sigma * Qinv(Pfa) / ts) / (P + sigma / ts));
};
```

```
inline double AC_detector(cvec x, int Td) {
    double sigma_sqr = (x * conj(x)).real() / (2 * x.length());
    cvec xl = x.left(x.length() - Td);
    cvec xr = x.right(x.length() - Td);
    return (xl * conj(xr)).real() / (2 * xl.length() * sigma_sqr);
};

inline double AC_Th(double Pfa, int M) {
    return Qinv(Pfa) / sqrt(2 * M);
};


inline double AC_Pd(double Pfa, double SNR, double CP, int M) {
    double eta_l = AC_Th(Pfa, M);
    double rho1 = CP * SNR / ((1 + CP)*(1 + SNR));
    return Qfunc(sqrt(2 * M)*(eta_l - rho1) / (1 - rho1 * rho1));
};


int simulate_ED() {
    string filename = "DVB-T_energy_detection_8_2_2_Pfa_0.1_T_0.01792_AWGN.it";
    cvec signal;        //The DVB-T  signal
    cvec w;             //White noise
    cvec x;             //Received signal
    AWGN_Channel awgn;
    ivec dvbt_vec = "8,2,2";
    ivec modp = "4 2";
    vec codep = "0.75 0.875";
    double T = 0.01792;
    vec dvbt_times, taul;
    ivec dvbt_iparams;
    cvec hl;
    double eta_energy;
    string type = "AWGN";
    string threshold = "theoretical";

    DVBT_parameters(dvbt_vec, dvbt_times, dvbt_iparams);
    Fading_Parameters(type, hl, taul);
    signal = SG_DVBT(dvbt_vec, modp, codep, hl, taul, T, 0, 0, 1);

    double Pfa = 0.1;
    int nrof_hits = 1000;
    int nrof_sims = 10 * ceil_i(nrof_hits / min(Pfa, 1 - Pfa));
    int eta_ind = nrof_sims - floor_i(Pfa * nrof_sims) - 1;
    int max_sim = 20 * nrof_hits;

    vec snr_dB = linspace(-40, -18, 12);
    vec Pd_energy = zeros(length(snr_dB));
    vec Pd_energy_theoretical = zeros(length(snr_dB));

    for (int snr_ind = 0; snr_ind < length(snr_dB); snr_ind++) {
        cout << "Now working on SNR " << snr_dB(snr_ind) << " dB" << endl;
        double sigma = Ps * dvbt_times(0) / inv_dB(snr_dB(snr_ind));
        awgn.set_noise(sigma);
```

```
        if (threshold == "simulate") {
            vec llr_noise_energy = zeros(nrof_sims);
            for (int sim = 0; sim < nrof_sims; sim++) {
                w = awgn(zeros_c(signal.length()));
                llr_noise_energy(sim) = Power(w, dvbt_times(0));
            }
            sort(llr_noise_energy);
            eta_energy = llr_noise_energy(eta_ind);
        }

        if (threshold == "theoretical") {
            eta_energy = Power_Th(Pfa, sigma, signal.length(), dvbt_times(0));
        }
        cout << "Thresholds have been computed " << endl;
        cout << "The threshold is:" << eta_energy << endl;

        int hits = 0;
        double llr_signal_energy;
        int energy_detect = 0;
        int energy_miss = 0;
        int sims = 0;

        while ((hits < nrof_hits) && (sims < max_sim)) {
            w = awgn(zeros_c(signal.length()));
            x = signal + w;
            sims++;
            llr_signal_energy = Power(x, dvbt_times(0));
            llr_signal_energy > eta_energy ? energy_detect++ : energy_miss++;
            hits = min(energy_miss, energy_detect);
        }
        cout << sims << " simulations needed" << endl;
        Pd_energy(snr_ind) = (double) energy_detect / sims;
        Pd_energy_theoretical(snr_ind) = 1 - Power_Pmd(Pfa, Ps, sigma, signal.length(), dvbt_times(0));
        cout << "Probability of detection: " << Pd_energy(snr_ind) << endl;
        cout << "Theoretical probability of detection: " << Pd_energy_theoretical(snr_ind) << endl;
    }

    //Save the results to file:
    it_file ff;
    ff.open(filename);
    ff << Name("B") << dvbt_vec(0);
    ff << Name("mode") << dvbt_vec(1);
    ff << Name("CP") << pow(2, -dvbt_vec(2));
    ff << Name("snr_dB_ED") << snr_dB;
    ff << Name("Pfa") << Pfa;
    ff << Name("Pd_energy") << Pd_energy;
    ff << Name("Pd_energy_theoretical") << Pd_energy_theoretical;
    ff.close();

    cout << "Results saved to: " << filename << endl;
    return 0;
}
```

```
int simulate_AC() {
    string filename = "DVB-T_AC_detection_8_2_2_Pfa_0.1_T_0.01792_AWGN.it";
    cvec signal;        // The DVB-T signal
    cvec w;             // White noise
    cvec x;             // Received signal
    AWGN_Channel awgn;
    ivec dvbt_vec = "8,2,2";
    ivec modp = "4 2";
    vec codep = "0.75 0.875";
    double T = 0.01792;
    vec dvbt_times, taul;
    ivec dvbt_iparams;
    cvec hl;
    double eta_ac;
    string type = "AWGN";
    string threshold = "theoretical";

    DVBT_parameters(dvbt_vec, dvbt_times, dvbt_iparams);
    Fading_Parameters(type, hl, taul);
    signal = SG_DVBT(dvbt_vec, modp, codep, hl, taul, T, 0, 0, 1);

    double Pfa = 0.1;
    int nrof_hits = 1000;
    int max_sim = 20 * nrof_hits;
    int nrof_sims = 10 * ceil_i(nrof_hits / min(Pfa, 1 - Pfa));
    int eta_ind = nrof_sims - floor_i(Pfa * nrof_sims) - 1;

    vec snr_dB = linspace(-32, -14, 10);
    vec Pd_ac = zeros(length(snr_dB));
    vec Pd_ac_theoretical = zeros(length(snr_dB));

    for (int snr_ind = 0; snr_ind < length(snr_dB); snr_ind++) {
        cout << "Now working on SNR " << snr_dB(snr_ind) << " dB" << endl;

        double sigma = Ps * dvbt_times(0) / inv_dB(snr_dB(snr_ind));
        awgn.set_noise(sigma);

        if (threshold == "simulate") {
            vec llr_noise_ac = zeros(nrof_sims);
            for (int sim = 0; sim < nrof_sims; sim++) {
                w = awgn(zeros_c(signal.length()));
                llr_noise_ac(sim) = AC_detector(w, 1024 * dvbt_vec(1));
            }
            sort(llr_noise_ac);
            eta_ac = llr_noise_ac(eta_ind);
        }

        if (threshold == "theoretical") {
            eta_ac = AC_Th(Pfa, signal.length() - 1024 * dvbt_vec(1));
        }

        cout << "Thresholds have been computed " << endl;
        cout << "The threshold is:" << eta_ac << endl;

        int hits = 0;
```

```
        double llr_signal_ac;
        int ac_detect = 0;
        int ac_miss = 0;
        int sims = 0;

        while ((hits < nrof_hits) && (sims < max_sim)) {
            w = awgn(zeros_c(signal.length()));
            x = signal + w;
            sims++;
            llr_signal_ac = AC_detector(x, 1024 * dvbt_vec(1));
            llr_signal_ac > eta_ac ? ac_detect++ : ac_miss++;
            hits = min(ac_miss, ac_detect);
        }

        cout << sims << " simulations needed" << endl;

        Pd_ac(snr_ind) = (double) ac_detect / sims;
        Pd_ac_theoretical(snr_ind) = AC_Pd(Pfa, inv_dB(snr_dB(snr_ind)), pow(2, -dvbt_vec(2)), signal.length() -
        1024 * dvbt_vec(1));
        cout << "Probability of detection: " << Pd_ac(snr_ind) << endl;
        cout << "Theoretical probability of detection: " << Pd_ac_theoretical(snr_ind) << endl;
    }

    //Save the results to file:
    it_file ff;
    ff.open(filename);
    ff << Name("B") << dvbt_vec(0);
    ff << Name("mode") << dvbt_vec(1);
    ff << Name("CP") << pow(2, -dvbt_vec(2));
    ff << Name("snr_dB_ac") << snr_dB;
    ff << Name("Pfa") << Pfa;
    ff << Name("Pd_ac") << Pd_ac;
    ff << Name("Pd_ac_theoretical") << Pd_ac_theoretical;
    ff.close();

    cout << "Results saved to: " << filename << endl;
    return 0;
}
```

```
int simulate_AC_CP() {
    string filename = "DVB-T_AC_CP_detection_8_2_2_Pfa_0.1_T_0.0005_AWGN.it";
    cvec signal; //The DVB-T  signal
    cvec w; //White noise
    cvec x; //Received signal
    AWGN_Channel awgn;
    ivec dvbt_vec = "8,2,2";
    ivec modp = "4 2";
    vec codep = "0.75 0.875";
    double T = 0.00112;
    vec dvbt_times, taul;
    ivec dvbt_iparams;
    cvec hl;
    double eta_ac_cp, eta_ac_cpR;
    string type = "AWGN";
```

```
string threshold = "simulate";

DVBT_parameters(dvbt_vec, dvbt_times, dvbt_iparams);
Fading_Parameters(type, hl, taul);
signal = SG_DVBT(dvbt_vec, modp, codep, hl, taul, T, 0, 0, 1);

double Pfa = 0.05;
int nrof_hits = 20;
int max_sim = 20 * nrof_hits;
int nrof_sims = 10 * ceil_i(nrof_hits / min(Pfa, 1 - Pfa));
int eta_ind = nrof_sims - floor_i(Pfa * nrof_sims) - 1;

vec snr_dB = linspace(-30, -30, 1);
vec Pd_ac_cp = zeros(length(snr_dB));
vec Pd_ac_cpR = zeros(length(snr_dB));

for (int snr_ind = 0; snr_ind < length(snr_dB); snr_ind++) {
    cout << "Now working on SNR " << snr_dB(snr_ind) << " dB" << endl;

    double sigma = Ps * dvbt_times(0) / inv_dB(snr_dB(snr_ind));
    awgn.set_noise(sigma);

    if (threshold == "simulate") {
        vec llr_noise_ac_cp = zeros(nrof_sims);
        vec llr_noise_ac_cpR = zeros(nrof_sims);
        for (int sim = 0; sim < nrof_sims; sim++) {
            w = awgn(zeros_c(signal.length()));
            llr_noise_ac_cpR(sim) = CP_detector(w, dvbt_vec(1) * pow(2, 10 - dvbt_vec(2)), 1024 *
            dvbt_vec(1));
            llr_noise_ac_cp(sim) = CP_detectorA(w, dvbt_vec(1) * pow(2, 10 - dvbt_vec(2)), 1024 *
            dvbt_vec(1));
        }
        sort(llr_noise_ac_cp);
        sort(llr_noise_ac_cpR);
        eta_ac_cp = llr_noise_ac_cp(eta_ind);
        eta_ac_cpR = llr_noise_ac_cpR(eta_ind);
    }

    if (threshold == "theoretical") {
        double eta_ac_cp = 0;
    }

    cout << "Thresholds have been computed " << endl;
    cout << "The threshold is:" << eta_ac_cp << endl;
    cout << "The threshold (real case) is:" << eta_ac_cpR << endl;

    double llr_signal_ac_cp;
    double llr_signal_ac_cpR;
    int ac_cp_detect = 0;
    int ac_cp_miss = 0;
    int ac_cp_detectR = 0;
    int ac_cp_missR = 0;
    int hits = 0;
    int sims = 0;
```

```
        while ((hits < nrof_hits) && (sims < max_sim)) {
            w = awgn(zeros_c(signal.length()));
            x = signal + w;
            sims++;
            llr_signal_ac_cpR = CP_detector(x, dvbt_vec(1) * pow(2, 10 - dvbt_vec(2)), 1024 * dvbt_vec(1));
            llr_signal_ac_cpR > eta_ac_cpR ? ac_cp_detectR++ : ac_cp_missR++;
            hits = min(ac_cp_missR, ac_cp_detectR);
        }

        cout << sims << " simulations needed" << endl;
        Pd_ac_cpR(snr_ind) = (double) ac_cp_detectR / sims;
        cout << "Probability of detection (real case): " << Pd_ac_cpR(snr_ind) << endl;
        sims = 0;
        hits = 0;

        while ((hits < nrof_hits) && (sims < max_sim)) {
            w = awgn(zeros_c(signal.length()));
            x = signal + w;
            sims++;
            llr_signal_ac_cp = CP_detectorA(x, dvbt_vec(1) * pow(2, 10 - dvbt_vec(2)), 1024 * dvbt_vec(1));
            llr_signal_ac_cp > eta_ac_cp ? ac_cp_detect++ : ac_cp_miss++;
            hits = min(ac_cp_miss, ac_cp_detect);
        }

        cout << sims << " simulations needed" << endl;
        Pd_ac_cp(snr_ind) = (double) ac_cp_detect / sims;
        cout << "Probability of detection: " << Pd_ac_cp(snr_ind) << endl;
    }

    //Save the results to file:
    it_file ff;
    ff.open(filename);
    ff << Name("B") << dvbt_vec(0);
    ff << Name("mode") << dvbt_vec(1);
    ff << Name("CP") << pow(2, -dvbt_vec(2));
    ff << Name("snr_dB_AC_CP") << snr_dB;
    ff << Name("Pfa") << Pfa;
    ff << Name("Pd_ac_cp") << Pd_ac_cp;
    ff << Name("Pd_ac_cpR") << Pd_ac_cpR;
    ff.close();

    cout << "Results saved to: " << filename << endl;
    return 0;
}


int simulate_Pilot() {
    string filename = "DVB-T_Pilot_detection_8_2_2_Pfa_0.05.it";
    ivec dvbt_vec = "8,2,2";
    vec dvbt_times;
    ivec dvbt_iparams;
    double Pfa = 0.05;
    int nrof_hits = 1000;
    int N = 1;          // nr of symbols/4;
```

```
vec snr_dB = linspace(-40, -20, 11);

cvec m;              // Pilot signal
cvec s;              // OFDM signal
cvec w;              // White noise
cvec x;              // Received signal
cvec q;              // Frequency domain OFDM symbol data
cmat cmlk_mat;  // Frequency domain data
AWGN_Channel awgn;
OFDM ofdm;
ivec modp = "4 4";
vec codep = "0.75 0.875";
vec Pd_pilot = zeros(length(snr_dB));
vec Pd_pilotA = zeros(length(snr_dB));

DVBT_parameters(dvbt_vec, dvbt_times, dvbt_iparams);
int Nfft = pow(2, 10) * dvbt_vec(1);
int Ncp = pow(2, 10 - dvbt_vec(2)) * dvbt_vec(1);
ofdm.set_parameters(Nfft, Ncp, 1);
double Ps = ((16.0 / 9) + dvbt_vec(1)*(1529 + 175 * 16.0 / 9) / 2) / (dvbt_times(0) * dvbt_iparams(3));
int nrof_sims = 1 * ceil_i(nrof_hits / min(Pfa, 1 - Pfa));
int eta_ind = nrof_sims - floor_i(Pfa * nrof_sims) - 1;
cmlk_mat = cmlk_pilots(dvbt_vec);
m.set_size(0);

for (int n = 0; n < 4; n++) {
    q = cmlk_mat.get_row(n);
    m = concat(m, ofdm.modulate(concat(q.right(1 + dvbt_iparams(1) / 2), zeros_c(Nfft - q.length()),
    q.left(dvbt_iparams(1) / 2))));          //OFDM-signal
}

for (int snr_ind = 0; snr_ind < length(snr_dB); snr_ind++) {
    cout << "Now working on SNR " << snr_dB(snr_ind) << " dB" << endl;
    double sigma = Ps * dvbt_times(0) / inv_dB(snr_dB(snr_ind));
    awgn.set_noise(sigma);
    vec llr_noise_pilot = zeros(nrof_sims);
    vec llr_noise_pilotA = zeros(nrof_sims);

    for (int sim = 0; sim < nrof_sims; sim++) {
        w = awgn(zeros_c(4 * N * dvbt_iparams(3)));
        llr_noise_pilot(sim) = Pilot_detector(w, m);
        llr_noise_pilotA(sim) = Pilot_detectorA(w, m);
    }

    sort(llr_noise_pilot);
    sort(llr_noise_pilotA);
    double eta_energy = llr_noise_pilot(eta_ind);
    double eta_energyA = llr_noise_pilotA(eta_ind);

    cout << "Thresholds have been computed " << endl;
    cout << "The threshold is:" << eta_energy << endl;
    cout << "The threshold is:" << eta_energyA << endl;

    int hits = 0;
    double llr_signal_pilot;
```

```
    double llr_signal_pilotA;

    int ac_detect = 0;
    int ac_miss = 0;
    int ac_detectA = 0;
    int ac_missA = 0;
    int max_sim = 20 * nrof_hits;
    int symbol_nr = 0;
    int sims = 0;
    cmlk_mat = cmlk(0, dvbt_vec, modp, codep);

    while (hits < nrof_hits // ||  sims < 1000 ) {
        s.set_size(0);
        for (int n = 0; n < 4 * N; n++) {
            if (symbol_nr == 68) {
                symbol_nr = 0;
                cmlk_mat = cmlk(0, dvbt_vec, modp, codep);
            }
            q = cmlk_mat.get_row(symbol_nr);
            symbol_nr++;
            s = concat(s, ofdm.modulate(concat(q.right(1 + dvbt_iparams(1) / 2), zeros_c(Nfft - q.length()),
            q.left(dvbt_iparams(1) / 2)))); //OFDM-signal
        }

      w = awgn(zeros_c(4 * N * dvbt_iparams(3)));
      x = s + w;
      sims++;
      llr_signal_pilot = Pilot_detector(x, m);
      llr_signal_pilotA = Pilot_detectorA(x, m);
      llr_signal_pilot > eta_energy ? ac_detect++ : ac_miss++;
      llr_signal_pilotA > eta_energyA ? ac_detectA++ : ac_missA++;
      hits = min(ac_miss, ac_detect);

        if (sims >= max_sim) {
            break;
        }
    }

    cout << sims << " simulations needed" << endl;

    Pd_pilot(snr_ind) = (double) ac_detect / sims;
    Pd_pilotA(snr_ind) = (double) ac_detectA / sims;
    cout << "Probability of detection: " << Pd_pilot(snr_ind) << endl;
    cout << "Probability of detectionA: " << Pd_pilotA(snr_ind) << endl;
}

//Save the results to file:
it_file ff;
ff.open(filename);
ff << Name("B") << dvbt_vec(0);
ff << Name("mode") << dvbt_vec(1);
ff << Name("CP") << pow(2, -dvbt_vec(2));
ff << Name("snr_dB_pilot") << snr_dB;
ff << Name("Pfa") << Pfa;
ff << Name("Pd_pilot") << Pd_pilot;
```

```
    ff << Name("Pd_pilotA") << Pd_pilotA;
    ff.close();

    cout << "Results saved to: " << filename << endl;
    return 0;
}


int simulate_CP_Pilot() {
    string filename = "DVB-T_CP_Pilot_detection_8_2_2_Pfa_0.05.it";
    ivec dvbt_vec = "8,2,2";
    vec dvbt_times;
    ivec dvbt_iparams;
    double Pfa = 0.05;
    int nrof_hits = 1000;
    int N = 1;                  // nr of symbols/4;

    vec snr_dB = linspace(-40, -16, 13);
    cvec m;                     // Pilot signal
    cvec s;                     // OFDM signal
    cvec w;                     // White noise
    cvec x;                     // Received signal
    cvec q;                     // Frequency domain OFDM symbol data
    cmat cmlk_mat;              // Frequency domain data
    AWGN_Channel awgn;
    OFDM ofdm;
    ivec modp = "4 4";
    vec codep = "0.75 0.875";
    vec Pd_CP_pilot = zeros(length(snr_dB));
    vec Pd_CP_pilotA = zeros(length(snr_dB));

    DVBT_parameters(dvbt_vec, dvbt_times, dvbt_iparams);
    int Nfft = pow(2, 10) * dvbt_vec(1);
    int Ncp = pow(2, 10 - dvbt_vec(2)) * dvbt_vec(1);
    ofdm.set_parameters(Nfft, Ncp, 1);
    double Ps = ((16.0 / 9) + dvbt_vec(1)*(1529 + 175 * 16.0 / 9) / 2) / (dvbt_times(0) * dvbt_iparams(3));
    int nrof_sims = 1 * ceil_i(nrof_hits / min(Pfa, 1 - Pfa));
    int eta_ind = nrof_sims - floor_i(Pfa * nrof_sims) - 1;
    cmlk_mat = cmlk_pilots(dvbt_vec);
    m.set_size(0);

    for (int n = 0; n < 4; n++) {
        q = cmlk_mat.get_row(n);
        m = concat(m, ofdm.modulate(concat(q.right(1 + dvbt_iparams(1) / 2), zeros_c(Nfft - q.length()),
        q.left(dvbt_iparams(1) / 2)))); //OFDM-signal
    }

    for (int snr_ind = 0; snr_ind < length(snr_dB); snr_ind++) {
        cout << "Now working on SNR " << snr_dB(snr_ind) << " dB" << endl;
        double sigma = Ps * dvbt_times(0) / inv_dB(snr_dB(snr_ind));
        awgn.set_noise(sigma);
        vec llr_noise_CP_pilot = zeros(nrof_sims);
        vec llr_noise_CP_pilotA = zeros(nrof_sims);

        for (int sim = 0; sim < nrof_sims; sim++) {
```

```
    w = awgn(zeros_c(4 * N * dvbt_iparams(3)));
    llr_noise_CP_pilot(sim) = CP_pilot_detector(w, m, Ncp, Nfft);
    llr_noise_CP_pilotA(sim) = CP_pilot_detectorA(w, m, Ncp, Nfft);
}

sort(llr_noise_CP_pilot);
sort(llr_noise_CP_pilotA);
double eta_energy = llr_noise_CP_pilot(eta_ind);
double eta_energyA = llr_noise_CP_pilotA(eta_ind);

cout << "Thresholds have been computed " << endl;
cout << "The threshold is:" << eta_energy << endl;
cout << "The threshold is:" << eta_energyA << endl;

int hits = 0;
double llr_signal_CP_pilot;
double llr_signal_CP_pilotA;
int ac_detect = 0;
int ac_miss = 0;
int ac_detectA = 0;
int ac_missA = 0;
int max_sim = 20 * nrof_hits;
int symbol_nr = 0;
int sims = 0;
cmlk_mat = cmlk(0, dvbt_vec, modp, codep);

while (hits < nrof_hits // ||  sims < 1000 ) {
    s.set_size(0);
    for (int n = 0; n < 4 * N; n++) {
        if (symbol_nr == 68) {
            symbol_nr = 0;
            cmlk_mat = cmlk(0, dvbt_vec, modp, codep);
        }
        q = cmlk_mat.get_row(symbol_nr);
        symbol_nr++;
        s = concat(s, ofdm.modulate(concat(q.right(1 + dvbt_iparams(1) / 2), zeros_c(Nfft - q.length()),
        q.left(dvbt_iparams(1) / 2)))); //OFDM-signal
    }
    w = awgn(zeros_c(4 * N * dvbt_iparams(3)));
    x = s + w;
    sims++;
    llr_signal_CP_pilot = CP_pilot_detector(x, m, Ncp, Nfft);
    llr_signal_CP_pilotA = CP_pilot_detectorA(x, m, Ncp, Nfft);
    llr_signal_CP_pilot > eta_energy ? ac_detect++ : ac_miss++;
    llr_signal_CP_pilotA > eta_energyA ? ac_detectA++ : ac_missA++;
    hits = min(ac_miss, ac_detect);

  if (sims >= max_sim) {
        break;
  }
}

cout << sims << " simulations needed" << endl;
Pd_CP_pilot(snr_ind) = (double) ac_detect / sims;
Pd_CP_pilotA(snr_ind) = (double) ac_detectA / sims;
```

```
        cout << "Probability of detection: " << Pd_CP_pilot(snr_ind) << endl;
        cout << "Probability of detectionA: " << Pd_CP_pilotA(snr_ind) << endl;
    }

    //Save the results to file:
    it_file ff;
    ff.open(filename);
    ff << Name("B") << dvbt_vec(0);
    ff << Name("mode") << dvbt_vec(1);
    ff << Name("CP") << pow(2, -dvbt_vec(2));
    ff << Name("snr_dB_CP_pilot") << snr_dB;
    ff << Name("Pfa") << Pfa;
    ff << Name("Pd_CP_pilot") << Pd_CP_pilot;
    ff << Name("Pd_CP_pilotA") << Pd_CP_pilotA;
    ff.close();

    cout << "Results saved to: " << filename << endl;
    return 0;
}
```